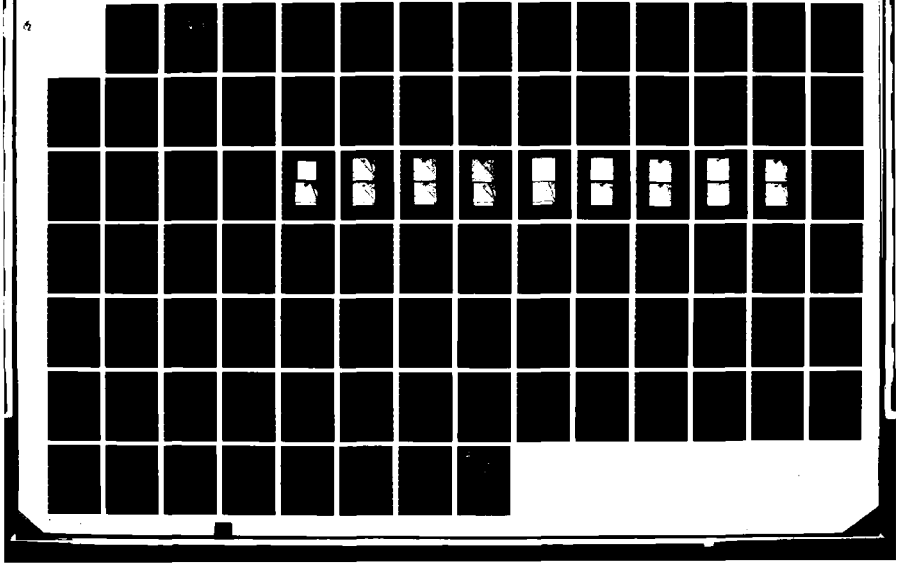
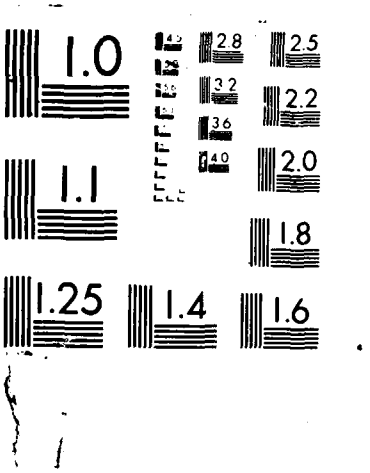
AD-A189 340 A TEXTURE ANALYSIS APPROACH TO COMPUTER VISION FOR IDENTIFICATION OF ROADS IN AERIAL PHOTOGRAPHS(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA J M SANDO DEC 87 1/1

UNCLASSIFIED F/G 8/2 NL




1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

NPS52-87-052

# NAVAL POSTGRADUATE SCHOOL

## Monterey, California




# THESIS

A TEXTURE ANALYSIS APPROACH
TO COMPUTER VISION FOR
IDENTIFICATION OF ROADS IN AERIAL
PHOTOGRAPHS

by

Jean M. Sando

December 1987

Thesis Advisor: Robert B. McGhee



Prepared for:
USACDEC
Ft. Ord, California 93941

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral R. C. Austin
Superintendent

Kneale T. Marshall
Acting Provost

This thesis is prepared in conjunction with research sponsored in part by contract from the United States Army Combat Developments Experimentation Center (USACDEC) under MIPR ATEC 88-86.



The issuance of this thesis as a technical report is concurred by:

ROBERT B. MCGHEE
Professor
of Computer Science

Reviewed by:

VINCENT Y. LUM
Chairman
Department of Computer Science

Released by:

JAMES M. FREMGEN
Acting Dean of Information and
Policy Science

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

# REPORT DOCUMENTATION PAGE

| | |
|---|---|
| 1a. REPORT SECURITY CLASSIFICATION<br>Unclassified | 1b RESTRICTIVE MARKINGS |
| 2a. SECURITY CLASSIFICATION AUTHORITY | 3. DISTRIBUTION/AVAILABILITY OF REPORT<br>Approved for public release; Distribution is unlimited. |
| 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE | |
| 4. PERFORMING ORGANIZATION REPORT NUMBER(S)<br>NPS52-87-052 | 5. MONITORING ORGANIZATION REPORT NUMBER(S) |
| 6a. NAME OF PERFORMING ORGANIZATION<br>Naval Postgraduate School<br>6b. OFFICE SYMBOL (If applicable)<br>Code 52 | 7a. NAME OF MONITORING ORGANIZATION<br>Naval Postgraduate School |
| 6c. ADDRESS (City, State, and ZIP Code)<br>Monterey, California 93943-5000 | 7b. ADDRESS (City, State, and ZIP Code)<br>Monterey, California 93943-5000 |
| 8a. NAME OF FUNDING/SPONSORING ORGANIZATION<br>USACDEC<br>8b. OFFICE SYMBOL (If applicable) | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER<br>ATEC 88-86 |
| 8c. ADDRESS (City, State, and ZIP Code)<br>Ft. Ord, California 93941 | 10. SOURCE OF FUNDING NUMBERS<br>PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO |

11. TITLE (Include Security Classification)
A TEXTURE ANALYSIS APPROACH TO COMPUTER VISION FOR IDENTIFICATION OF ROADS IN AERIAL PHOTOGRAPHS (u)

12. PERSONAL AUTHOR(S)
Sando, Jean M.

| 13a. TYPE OF REPORT | 13b. TIME COVERED | 14. DATE OF REPORT (Year, Month, Day) | 15 PAGE COUNT |
|---|---|---|---|
| Master's Thesis | FROM _____ TO _____ | 1987 December | 88 |

16. SUPPLEMENTARY NOTATION

| 17. COSATI CODES | | | 18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number) |
|---|---|---|---|
| FIELD | GROUP | SUB-GROUP | Computer vision; texture |
| | | | |
| | | | |

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The development of computer vision and the identification of objects in an image is important for many areas of scientific, medical, and commercial ventures. The available literature details many experiments and algorithms in these fields. In this study, the possibility of identifying road surfaces in aerial black and white photographs using texture analysis is examined. A system using texture that was successful in identifying road surfaces is presented as part of this research.

| | |
|---|---|
| 20 DISTRIBUTION/AVAILABILITY OF ABSTRACT<br>☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS | 21. ABSTRACT SECURITY CLASSIFICATION<br>Unclassified |
| 22a. NAME OF RESPONSIBLE INDIVIDUAL<br>Prof. Robert B. McGhee | 22b TELEPHONE (Include Area Code)<br>(408) 646-2095<br>22c OFFICE SYMBOL<br>Code 52Mz |

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Approved for public release; distribution is unlimited.

# A Texture Analysis Approach to Computer Vision for Identification of Roads in Aerial Photographs

by

Jean M. Sando
Lieutenant, United States Navy
B.S., University of Florida, 1979

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1987

Author: _____
Jean M. Sando

Approved by: _____
Robert B. McGhee, Thesis Advisor

_____
Michael J. Zyda, Second Reader

_____
Vincent Y. Lum, Chairman,
Department of Computer Science

_____
James Fremgen,
Acting Dean of Information and Policy Sciences

## ABSTRACT

The development of computer vision and the identification of objects in an image is important for many areas of scientific, medical, and commercial ventures. The available literature details many experiments and algorithms in these fields. In this study, the possibility of identifying road surfaces in aerial black and white photographs using texture analysis is examined. A system using texture that was successful in identifying road surfaces is presented as part of this research.


TABLE OF CONTENTS

LIST OF TABLES

# LIST OF FIGURES

# I. INTRODUCTION

## A. BRIEF BACKGROUND AND PROBLEM STATEMENT

The development of computer vision is an ongoing area of research in the field of artificial intelligence. The ability of the computer to identify objects in images, or in the real world, opens up a whole new form of data input. Research in the field varies, from vision in controlled environments, such as industrial assembly lines, to identification of objects in uncontrolled, real world environments.

Vision research has been broken down into two phases. The first phase, early processing, has seen a great deal of experimentation in using the statistical properties of texture to identify regions in images [Ref. 1,2,3]. This research has produced many complicated mathematical equations to differentiate between textures [Ref. 1,2,3]. The purpose of this study is to identify two textures using relatively simple equations.

In the second phase of computer vision, actual objects are identified. This is accomplished by matching descriptions. This study does not go into the second phase of vision.

To achieve the desired results of this thesis, it is necessary to first have some way of storing an image in the computer. The author along with others developed a system for this purpose prior to this study [Ref. 4]. This system digitizes and displays images up to 1024 by 768 pixels.

For data, the U.S. Army provided aerial photographs of Fort Hunter Liggett, Ca. The objective of this thesis was to pick out road surfaces on these photographs.

## B. METHODOLOGY

In beginning this study, it was necessary to make a guess as to what qualities on-road and off-road regions possess to enable the human vision system to distinguish them. Once the qualities were determined, a way of representing those qualities to the computer was required. A mathematical strategy was then needed to represent them in a simple way. The next step was to process a series of test images to evaluate the equations. Finally, it was necessary to analyze the results of the test to determine if the equations worked, and which worked the best.

## C. ORGANIZATION

A brief review of the basics of computer vision and previous work is presented in Chapter Two. Chapter Three consists of a detailed problem formulation. Chapter Four details the implementation of the recognition system. Chapter Five is an evaluation of the results obtained in scanning an image using different methods. The final chapter, Chapter Six contains conclusions and recommendations for further work in this area.

# II. SURVEY OF PREVIOUS WORK

## A. INTRODUCTION

In this chapter, a brief background of previous work in computer vision is given. The emphasis is on texture analysis, with some description of early vision techniques and a brief description of higher level processing.

## B. GENERAL APPROACHES TO COMPUTER VISION

The general approach to computer vision is to break the process down into two distinct steps. In the first step, the goal is to get useful information from the original image. The second step tries to identify objects from the information found in the first step. Such processing is called early processing. In early processing, every part of the image is processed in parallel. Subsequent processing uses the data produced in early processing sequentially and extracts a symbolic description of the image. [Ref. 5: page 95]

The way in which computer vision is broken down is similar to human vision. Humans appear to make some discriminations in an image automatically and in parallel. These early discriminations include texture segregation. In the second stage of human vision, focused attention is required and this appears to be performed serially. This second stage identifies the objects we see. [Ref. 6: page 156]

### 1. Early Vision Image Processing

In early vision processing, the first step is a description of the edges and other interesting features of an image. This description is commonly called the *primal sketch.* To produce the primal sketch, it is necessary to analyze the image and abstract the *changes* in gray levels to produce a database. This database contains data structures, each describing a feature of the image. Each item in the image is made up of one or more edgelets, or small edges. Each feature of an image is a combinations of these edgelets and are categorized as follows: *edges*, which consist of edgelets with the same orientation, *bars*, which are two rows of edgelets going in the same direction, and *blobs*, which are edgelets enclosing a small region. [Ref. 5: pages 98-101]

After developing a primal sketch, there are several ways to extract more information. These processes are; *motion analysis*, which is identifying features that seem to be moving together, *stereo disparity*, which analyses the difference between the

two views to determine depth and orientation, *photometry*, which determines the shape of surfaces from shading and *grouping processing*, that organizes smaller features into larger ones [Ref. 5: page 99].

Two types of grouping processing that are particularly interesting to the work in this study are *virtual lines* and *texture analysis.* In the virtual lines technique, the computer draws connecting lines between blobs or endpoints. The decision as to which items to connect comes from a *histogram* that gives greater weigh to near objects. In texture analysis, the computer tries to identify repeated elements or patterns. There are several methods used to accomplish this and they are discussed below [Ref. 5: pages 111, 120].

### 2. Late Vision Symbolic Description

The second step in computer vision is scene description. This step is broken down into three stages. First, the data produced in the first step, is converted to a 3-D object-oriented stage. In the second stage, the computer compares the objects in the image with a library of known object types. The last stage is to match an object with a known object type, in this way identifying the object in the image. To accomplish stage one, a *shape notation* is required. A shape notation is a standard way of describing objects both in the library and in the image. In the third stage algorithms are required that take into consideration that an object in an image can be partially hidden, rotated, seen from a different angle, or that the object varies from the 'standard' object in the library [Ref. 5: pages 144-155].

## C. IMAGE SEGMENTATION

### 1. Types of Segmentation

There are many segmentation methods, with many variations. Three of these methods are split, merge, and split and merge. In the split method, a region is obtained from the image. This region is compared to known classes. If this region is identified as containing only one class, it labeled as that class. If not it is split and the process is repeated on each part until all parts are labeled. [Ref. 2: page 274]

In the merge method, a small area is chosen and classified. Then each surrounding area is examined. If an adjoining area is the same class, it is combined with the first area and the process is repeated. In the split and merge method, the split method is used until a region is classified. Then the merge method is applied [Ref. 2: page 274].

Another method is that of using regions called *local windows*. In this method, the image is divided up into many windows. The size and shape of these windows depends on the image and the size and shape of the objects to be identified. The size and shape of the windows is a critical factor in the process of correctly identifying the objects in the image [Ref. 1: page 368].

**2. Features for Classification**

The regions described above normally are classified in the early stages of vision processing using either gray levels or textures. Within the area of texture, there are two major analytical approaches, statistical and structural [Ref. 3: page 336]. The structural approach is used if the texture has a clear pattern and can be matched with stored patterns. The statistical approach is used when pattern matching is not possible.

There are many possible statistical approaches that have been tried, some with very good results. In a project at Louisiana State University, six texture statistical measurements were chosen [Ref. 2]. These measurements were: inertia, cluster shade, cluster prominence, local homogeneity, energy, and entropy. Their results of classifying aerial photographs into nine classes produced a result of 90% overall correct classification [Ref. 2: page 274].

## D. IMAGE SEMANTICS AND SYNTACTIC PATTERN RECOGNITION

In general,the syntactic features of roads are that they consist of two continuous edges with a separation not greater then X, and not less than Y. Another feature is that these edges or roads are connected; i.e. a road does not have two ends. Roads can have curves, but these curves have a radius no less than Z. Of course the values for X, Y, and Z depend on the scale of a particular given image.

## E. HARDWARE AND SOFTWARE SYSTEMS FOR IMAGE ACQUISITION AND PROCESSING

### 1. Cameras

#### *a. Pictures*

There are several media for obtaining images. These include slide transparencies, cine film, videotape and hard-copy representations such as photographs [Ref. 7: page 38]. Most of the work examined in this field uses black and white aerial photographs.

*b. Digitizing*

In order to get an image into the computer, it is scanned, digitized and stored in the computer pixel by pixel. Each pixel is identified by an x and y coordinate for its location and a gray level if a black and white photograph, or three numbers giving the red green and blue intensity of the pixel in the case of a color photograph.

2. **Displays**

Although it is not necessary for the computer in its processing to have a display device, it is preferable for humans, because of our limitations, to be able to see the results. There are many different graphics devices to choose from. The most common display device for interactive graphic is the raster scan, which can display a mono-color, gray level or color image.

3. **Languages and Computer Systems**

In most research, after the hardware is decided on there is a choice of several programming languages. These choices range from assembly languages to high level artificial intelligence languages such as Prolog and LISP. Of the languages available, Common LISP is particularly appealing because it is a DOD standard.

## F. SUMMARY

Much of the prior work in computer vision has been focused on the areas of early vision analysis, and in texture analysis. The way vision has been approached is parallel to the way it is believed that humans do vision processing. The use of texture has proven to be very successful in identifying regions of known classes. The identification of objects is achieved by creating descriptions of objects and then matching those descriptions to known objects. Although research in this field has met with success, there still remains much work to be done in both early and late vision processing.

# III. DETAILED PROBLEM FORMULATION

## A. INTRODUCTION

This chapter describes the methods considered in solving the problem of finding roads in aerial photographs, and the methods chosen for solving this problem. It includes the data format of the files produced by the digitizing camera on the VAX/VMS system, and the files produced on the IRIS system containing the scanned image road identification. Also described are the features selected for analysis and the utilization of these features in identification of on-road areas.

## B. DATA FORMAT

### 1. Digitized Image File

The map data consists of 500 by 500 pixel black and white digitized images. This data is contained in a file with a eighty-six byte header followed by the digitized image. The header consists of single byte character data and two byte integer data, with the least significant byte (LSB) first. The header consists of the following:

1. The integer 1, which indicates a black and white image.
2. An integer, which indicates the number of rows of pixels.
3. An integer, which indicates the number of columns of pixels.
4. Eighty characters which hold an optional comment.

The image data follows and is of type byte. [Ref. 4: page 15]

### 2. Scanned Image file

As an image is scanned to identify on-road areas, this data along with the data used in identification of these areas is stored in a file. The same file format is used for both types of scanning data files. These files consist of an eighty-eight byte header of general data, followed by the data that identifies the on-road areas. Since the version of LISP used (see Chapter Four) allows only storage of integers one byte in length, the integers and floating point numbers are broken down into two bytes of data. Table 1 shows the format of the first eighty-eight bytes of general data stored in the file.

The next item stored is a single byte integer, the size of the window. The rest of the file contains the data on the identified on-road areas. Each identified on-road area has five bytes of data stored. The first byte is an integer indicating the selected feature or features utilized in determining that the area is on-road. The next two bytes

TABLE 1

FILE FORMAT

| Field Name | Bytes |
|---|---|
| outer on road mean | 2 |
| inner on road mean | 2 |
| deviation of the outer on road mean | 3 |
| deviation of the inner on road mean | 3 |
| outer on road variance | 2 |
| inner on road variance | 2 |
| deviation of the outer on road variance | 3 |
| deviation of the inner on road variance | 3 |
| outer off road mean | 2 |
| inner off road mean | 2 |
| deviation of the outer off road mean | 3 |
| deviation of the inner off road mean | 3 |
| outer off road variance | 2 |
| inner off road variance | 2 |
| deviation of the outer off road variance | 3 |
| deviation of the inner off road variance | 3 |
| threshold of the outer mean | 3 |
| threshold of the inner mean | 3 |
| threshold of the outer variance | 3 |
| threshold of the inner variance | 3 |
| on road variance product | 3 |
| off road variance product | 3 |
| on road probability | 3 |
| off road probability | 3 |
| inner on road mean | 3 |
| variance on the inner on road mean | 3 |
| variance on the outer on road variance | 3 |
| variance on the inner on road variance | 3 |
| inner off road mean | 3 |
| variance on the inner off road mean | 3 |
| variance of the outer off road variance | 3 |
| variance of the inner off road variance | 3 |

are the x screen coordinate. The final two bytes are the y screen coordinate. These two byte integers are broken down as described above.

## C. SELECTED FEATURES

### 1. Moments

When using texture to identify image regions, there are two approaches to texture description: statistical and structural [Ref. 8: page 406]. The approach chosen for this work was statistical. In the statistical approach, there are many possible methods for extracting relative information about the textures. These methods include, but are not restricted to: means, weighted means, variance, weighted variances and more complete measurements such as inertia, local homogeneity and energy [Ref. 1,2: pages 368, 279]. The structural approach includes methods of pattern

matching using normalized correlation matching, and least-square error polynomial fits of image approximations [Ref. 1: page 369]. Since texture has meaning only when discussing a group of points or pixels, the image must be segmented into regions or areas. When discussing texture, these regions are referred to as *windows*. Again, there are several ways of dividing an image into windows. Windows can all be of uniform shape and size, or they can vary depending on prior analysis. In this study, all windows are square regions composed of between 4 and 225 pixels. Square windows were chosen for simplicity and the minimum and maximum size selected due to of the size of the roads on the maps used. The window that is being classified is defined as the *inner* window. The *outer* window refers to a region that is centered on the inner window, but is larger than the inner window. The outer window is used to help further classify the inner window.

The three features chosen for this study were the mean of the inner window, the variance of the inner window and the variance of the outer window. The mean of the outer window was not used because early tests indicated the results would be virtually identical to the result from the inner window mean. The mean of the inner window defines the gray level of the window. The gray level is important because in most cases there is a distinct difference in this feature between the two textures. The variance from the inner window defines the granularity, or roughness of the window. The outer window is used to determine if the granularity varies as the size of the window approaches or exceeds the size of the object, in this case the road. These moments were chosen because the calculations were relative simple and they represent features that intuitively suggest the different properties the human eye might pick out.

2. **Selecting Windows**

In order to detect on-road areas, the system must first be taught. The system is designed to distinguish between two type of textured areas. In order to instruct the system, the user must do several things. First, the user must decide on the size of the square inner and outer windows. The size of the windows depends on the size of the roads. The inner window should be smaller than the width of the roads and the outer window should be the of same size or larger than the width of the roads. The smaller window is the scan area being classified. When teaching the system, the inner window appears as a cursor to choose the on-road and off-road areas.

The user must also determine the number of areas that are used to instruct the system. This number should be large enough to include all different types of on-road and off-road areas.

## D. APPROACHES TO THRESHOLD SELECTION

### 1. Selected Features Considered Separately

As the image is scanned, one method to identify a window is to use each moment separately to determine the textures. In this first method, each scan window is tested against each of the three selected features, and results are stored indicating which of the features, if any, determine the scan window to be an on-road area. In order to test each feature, a threshold must be determined for each feature. Two methods to determine a threshold were tested. A simple method of picking the midpoint between on-road and off-road values was first used. This method produced good results.

A second method to chose a threshold, which proved more successful and is the method now used in the system, uses the normal distribution and density functions to compute an optimal threshold. This equation was derived as shown in Equations 3.1- 3.15 .

The Gaussian probability density function, also called the *normal curve*, is given by

$$p(x) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{\frac{-(x-\mu)^2}{2\sigma^2}} = N(\mu, \sigma) \qquad \text{(eqn 3.1)}$$

where $\mu$ is the mean value of x and $\sigma$ is the standard deviation of x. This function is graphically depicted in Figures 3.1 and 3.2

By definition, the shaded area under the left hand side of the curve in Figure 3.1 is given by

$$A = \int_{-\infty}^{t_x} N(\mu, \sigma)\, dx \qquad \text{(eqn 3.2)}$$

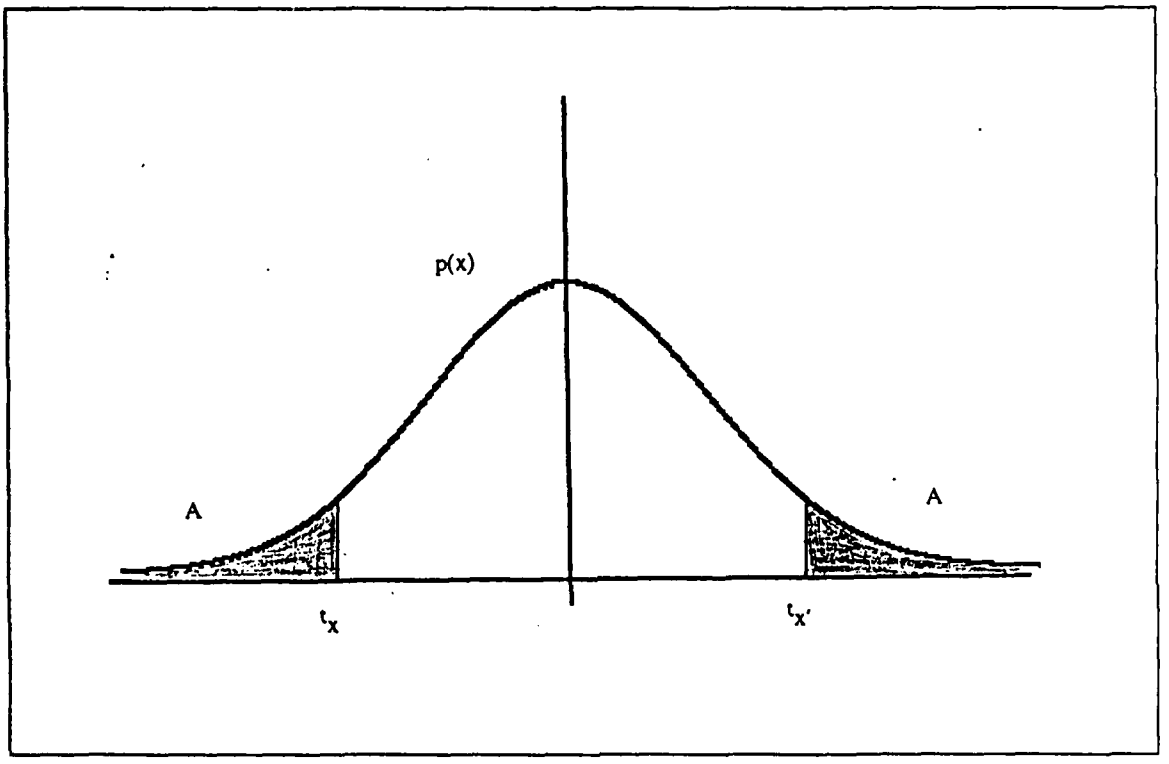



Figure 3.1 Normal Density Function p(x).

To determine A, it is necessary to standardize the Normal curve of N( $\mu$ , $\sigma$ ) so that it is in the form N(0,1). To accomplish this, let

$$y = \frac{x-\mu}{\sigma} \qquad \text{(eqn 3.3)}$$

Then,

$$y^2 = \frac{(x-\mu)^2}{\sigma^2} \qquad \text{(eqn 3.4)}$$

Substituting $y^2$ into Equation 3.1 and this result into Equation 3.2, it follows that

$$A = \int_{-\infty}^{t_y} \frac{1}{\sqrt{2\pi}} e^{\frac{-y^2}{2}} dy = \text{erf}(t_y) \qquad \text{(eqn 3.5)}$$

where erf(x) is the *error function* or cumulative gaussian probability density function. Substituting the upper limit of Equation 3.2 into Equation 3.3,

$$t_y = \frac{t_x - \mu}{\sigma} \qquad \text{(eqn 3.6)}$$

In the same manner as above, the area A under the right side of the curve in Figure 3.1 is given by

$$A = \int_{t_{x'}}^{\infty} N(\mu,\sigma)dx = \text{erf}(t_y) \qquad \text{(eqn 3.7)}$$

where the lower limit $t_{x'}$, is defined by

$$t_{x'} = t_x + 2(\mu - t_x) = 2\mu - t_x \qquad \text{(eqn 3.8)}$$

Solving Equation 3.7 for $t_x$,

$$t_x = 2\mu - t_{x'} \qquad \text{(eqn 3.9)}$$

Now substituting Equation 3.9 into Equation 3.6,

$$t_y = \frac{t_x - \mu}{\sigma} = \frac{\mu - t_{x'}}{\sigma} \qquad \text{(eqn 3.10)}$$

Now consider two distributions, $N(\mu_1, \sigma_1)$, and $N(\mu_2, \sigma_2)$, with a common threshold $t_x$. Referring to Figure 3.2, the areas under the curves is given by

$$A_1 = \text{erf}\left[\frac{\mu_1 - t_x}{\sigma_1}\right] \qquad \text{(eqn 3.11)}$$

$$A_2 = \text{erf}\left[\frac{t_x - \mu_2}{\sigma_2}\right] \qquad \text{(eqn 3.12)}$$

If the probability of classifying an on-road pixel as off-road is the same as classifying an off-road pixel as on-road, then $A_1 = A_2$ and thus, from Equation 3.11 and 3.12

$$\frac{(\mu_1 - t_x)}{\sigma_1} = \frac{(t_x - \mu_2)}{\sigma_2} \qquad \text{(eqn 3.13)}$$

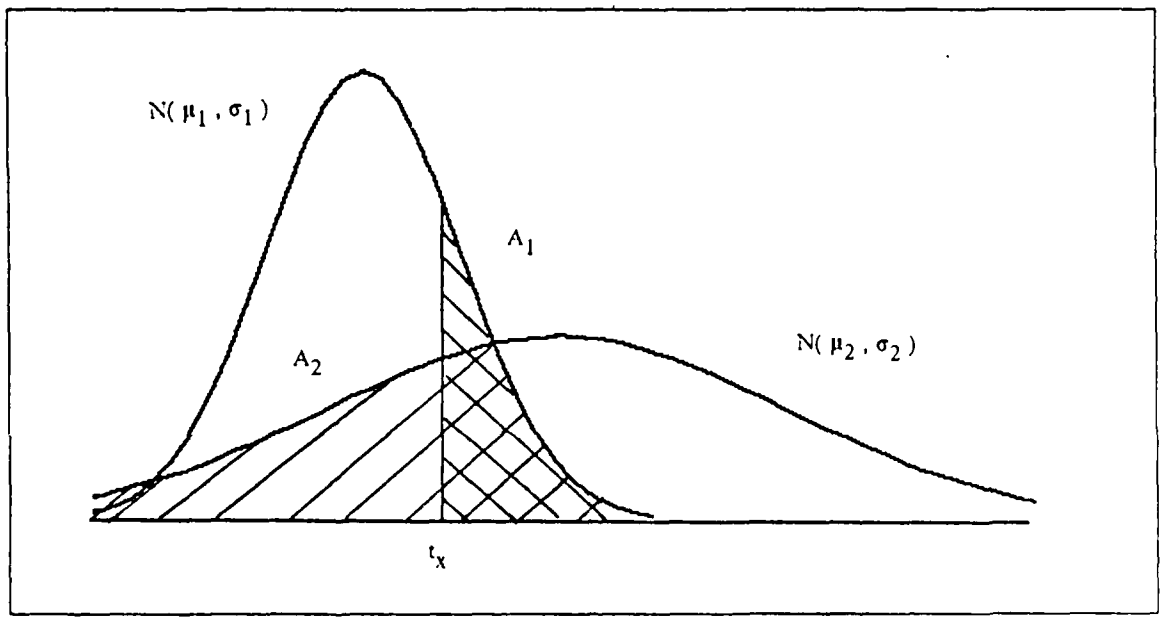



Figure 3.2 Normal Density Function of Two Gaussian Curves.

Equation 3.13 can be rewritten to obtain

$$(\sigma_1+\sigma_2)t_x = \sigma_2\mu_1+\sigma_1\mu_2 \qquad \text{(eqn 3.14)}$$

Thus, solving for $t_x$ from Equation 3.14, the final result for equal probability of these two types of error is

$$t_x = \frac{\sigma_1\mu_2+\sigma_2\mu_1}{\sigma_1+\sigma_2} \qquad \text{(eqn 3.15)}$$

[Ref. 9: pages 188-189]. In what follows, Equation 3.15 is the equation used to determine the individual thresholds of each of the three selected features.

2. **Selected Features Considered Together**

In this method of identifying a window, called a *Gaussian scan*, all three selected features are used together to determine if a region is on-road or off-road. In the methods described above, a classification based on a scalar measurement, a *threshold* was used for binary classification. In a more general classification based on vector measurements, a separating surface is needed instead of a simple threshold.

There are two ways to approach this problem. The first uses a hyperplane and a measure of the relative importance of the two types of error. For example, if both types of error are equally important, then a plane is found that cuts equal *volumes* off both tails of the total distribution curves. A problem with this approach is that in general it doesn't yield the minimum error. It yields the minimum error only in the special case of equal covariances for the Gaussian case [Ref. 10: page 194 Eq. 4].

A second approach that works in the more general case assumes that both types of error are equally important (1-0 loss function). In this approach,the optimal separating surface is a curved surface. The decision rule with respect to such a surface is to select class i such that i maximizes the function, f(i), given by

$$f(i) = \ln P_i - \ln\left[\prod_{j=1}^{L}\sigma_{ij}^2\right]^{\frac{1}{2}} - \frac{1}{2}\sum_{j=1}^{L}\frac{(x_j-m_{ij})^2}{\sigma_{ij}^2} \qquad \text{(eqn 3.16)}$$

where, j = {inner window mean, outer window variance, inner window variance }, and i = { on-road, off-road }. The variable $P_i$ is the prior probability that a scan window is on-road or off-road. The second term in Equation 3.16 is called the variance product and is the product of the variances of the mean of the inner window, the variance of the outer window and the variance of the inner window. The last term in this equation is a generalized distance from the center of the class distributions. This function produces a minimal error rate for both types of misclassification errors.

It should be noted that Equation 3.16 consists of all constants except for the $x_j$ . Thus, the optimal choice has the shortest generalized distance from the *class means.*

## E. EXPERIMENTAL FACILITIES

### 1. VAX/VMS System and the Digitizing Camera

Connected to the VAX/VMS System is the EIKONIX 785 Digitizer Camera System. The camera is capable of digitizing an image up to 4096 X 4096 pixels and storing this image in a file. The camera is capable of taking either a color or black and white image. The camera is equipped with a 55mm SLR lens with f-stop range of 2.8 to 22. The camera height is adjustable from 32 to 140 centimeters.

### 2. Display and Scanning System

The graphics system used is the Silicon Graphics Inc. IRIS Graphics Workstation. This system is capable of displaying images produced by the digitizing camera. The graphics station is limited to 1024 X 768 pixel display. Therefore, any image larger than this needs to be broken up. The images are displayed using RGB mode.

## F. SUMMARY

This project was designed to find features that would aid in identifying on-road areas in aerial photographs. Two different methods of determining a threshold were examined. These methods were a simple binary method and a method where the probability of both types of possible errors were made equal. Also presented was a Gaussian method of combining several features to produce a minimum probability of error.

# IV. IMPLEMENTATION OF THE RECOGNITION SYSTEM

## A. INTRODUCTION

This chapter contains descriptions of the major LISP and C functions used in the road identification system. It includes descriptions of the function purpose, input variables, the results of the function call, and any restrictions inherent in each function.

## B. MAJOR SOFTWARE LANGUAGE AND MODULES

### 1. Language

Although Common LISP was the language preferred by the author for image processing, it was not available on a suitable host at the time this study was undertaken. Therefore, in what follows, the earlier dialect, Franz LISP [Ref. 11] will be used. Since Franz LISP is largely upward compatible with Common LISP, this is not viewed as a serious limitation regarding further extensions of this study.

### 2. LISP Functions

#### *a. function INIT-FILE*

The purpose of this function is to display the digitized map image on the IRIS graphics display. It requires as input the name of the file containing the digitized image. It also requires the x and y screen coordinates of the upper left corner where the image is to be displayed. Although this function displays an image up to 1024 X 768 pixels, this system was designed to display two map images side by side. For this reason images should be no wider than 500 pixels.

The result of calling this function is a displayed image of the digitized map. This function is normally called once with screen coordinates (0,766) when the next function to be called is *main*. This displays the map image in the correct location so that the instructions displayed on the graphics screen do not overlap the map. This function is called twice to display two identical maps side by side with screen coordinates (0,766) and (524,766) when the next function to be run is *show-road*. *Show-road* has an x coordinate input of either 0 or 524.

The function *init-file* sets up the global variables *lines* and *cols*. Both of these values are read from the digitized map file. The C function *display* is the only function called from this function.

*b. function MAIN*

The purpose of this function is to instruct the system in the difference between the two textures of on-road and off-road. The required inputs are the outer and inner window sizes and the number of points for the teaching sample.

This LISP function is designed for interactive operator inputs. It displays a cursor the size of the inner window and requests that the operator choose the specified number of on-road and off-road areas. This function constructs a list of the sum of the points within each area, and from this list calculates the means and variances both on-road and off-road of the inner and outer windows. These values become global variables. This function also calculates the global variables of the variance product, used in the *Gaussian* function, the standard deviations and the thresholds for both the means and the variances.

Other functions called in this function are the LISP functions *a_window_mean*, *a_window_variance*, *mean*, *variance*, and *threshold*. The C functions called are *window1*, and *readsquare*.

*c. function SCAN-MAP*

The function *scan-map* scans the displayed image identifying on-road areas and stores these areas in two files. It is assumed that the map image is 500 X 500 pixels and is displayed at screen coordinates (0 766). The coordinates could be easily changed to input variables, but for the purposes of this study the use of the maximum square image was preferred. The inputs are the size of the outer and inner windows, the two file names for the output and the probability that any given area will be on-road. The two output files hold the results of the two methods used in identifying the scanned windows. The scan is conducted from left to right, top to bottom. If an area is identified as on-road, the lower right corner coordinate of the area is stored in the file, along with a code indicating which of the selected features was used to identify the area.

The LISP functions called from this program are *store-data*, *gaussian*, and *compare*. The C functions are *readsquare* and *writesquare*.

*d. functions STORE-DATA and READ-DATA*

The functions *store-data* and *read-data* are complements of each other. The function *store-data* stores all the global variables in the file given as the input parameter. This is normally called from *scan-map* and is used to store the global data in the file before the scan data. The function *read-data* is used to read the data out of the file before displaying the scan results. It is normally called from *show-road*.

*e. function SHOW-ROAD*

This function displays the results of the scan on the IRIS graphics workstation. The required inputs are the filename and the x screen coordinate at which the image is displayed. This function is designed to be called after *init-file* has been called to display the image. It assumes that the image was originally displayed and scanned with the y screen coordinate of 766.

## C. SUMMARY

The functions described in this chapter were designed to be called in a specific order. However they can all be called independently, within the restrictions given and with the required global variables set. Once an image is scanned, and the results saved, the file can be displayed at any time, with or without superimposing it over the map image. If the system is taught on one map image, and there are several images to be scanned that the operator feels contain the same textures of the same size, the system need only be taught on the first map image, and this data can be used for all similar images, by first reading out of a file the global variables.

# V. EVALUATION

## A. INTRODUCTION

To evaluate the system, forty-eight different experiments were conducted. These experiments were broken down into four groups, depending on how the off-road regions were selected when teaching the system. Table 2 shows the breakdown of regions. Each group was broken down into two subgroups, a and b. Subgroup a was conducted with an outer window size of 10 X 10 and an inner window size of 5 X 5. Subgroup b had an outer window size of 9 X 9 and an inner window size of 3 X 3.

All on-road regions chosen to instruct the system came from representative portions of both major roads shown in Figure 5.1 Specifically, the major roads are the vertical road and the road at approximately 20° from the horizonal.

TABLE 2
GROUPS FOR TEACHING OFF-ROAD REGIONS

| GROUP | TYPE REGIONS |
|---|---|
| 1 | 15 very dark |
| 2 | 3 very dark, 4 dark, 6 light, 2 very light |
| 3 | 15 light |
| 4 | 15 very light |

## B. 7 COLOR SCAN

In this scan, all three selected features inner window mean, inner window variance, and outer window variance were individually tested. The threshold was chosen for each feature as shown in Equation 3.15 . The results were color coded (see Table 3) and displayed.

The results from the first three groups of off-road regions, and subgroups from both sizes of windows were similar. The results can be seen in Figures 5.2, 5.3 and 5.4 . These figures show that the major roads were found, with red indicating all three features were used. The last test, as seen in Figure 5.5, produced different results. When the larger windows were used, the inner variance picked out the road, the outer

TABLE 3

7 COLOR SCAN COLOR CODES

| | |
|---|---|
| green | inner mean |
| blue | inner variance |
| yellow | outer variance |
| cyan | inner mean and inner variance |
| purple | inner variance and outer variance |
| brown | inner mean and outer variance |
| red | inner mean outer variance and inner variance |

variance picked out the off-road and the mean picked out the entire image. Using the smaller windows, the mean picked out the road, while both variances picked out the off-road. In this map image, this result indicates that when choosing off-road regions to be very light, the difference in the mean for the larger windows was reduced to near zero. With the smaller windows, the mean was the only feature that gave good results. It is clear that the way in which off-road regions were taught, and to a lesser extent, the size of the windows, has a great deal of influence on the results in this type of scan.

## C. GAUSSIAN SCAN

The Gaussian Scan is accomplished using all three selected features together as shown in Equation 3.16 . Forty experiments were run, using five different probabilities for a region being on-road. The four different groups given in Table 2 were used, as were the two different size of windows. Five gaussian scans were run in each subgroup using *a priori* road probabilities of .1, .3, .5, .7 and .9.

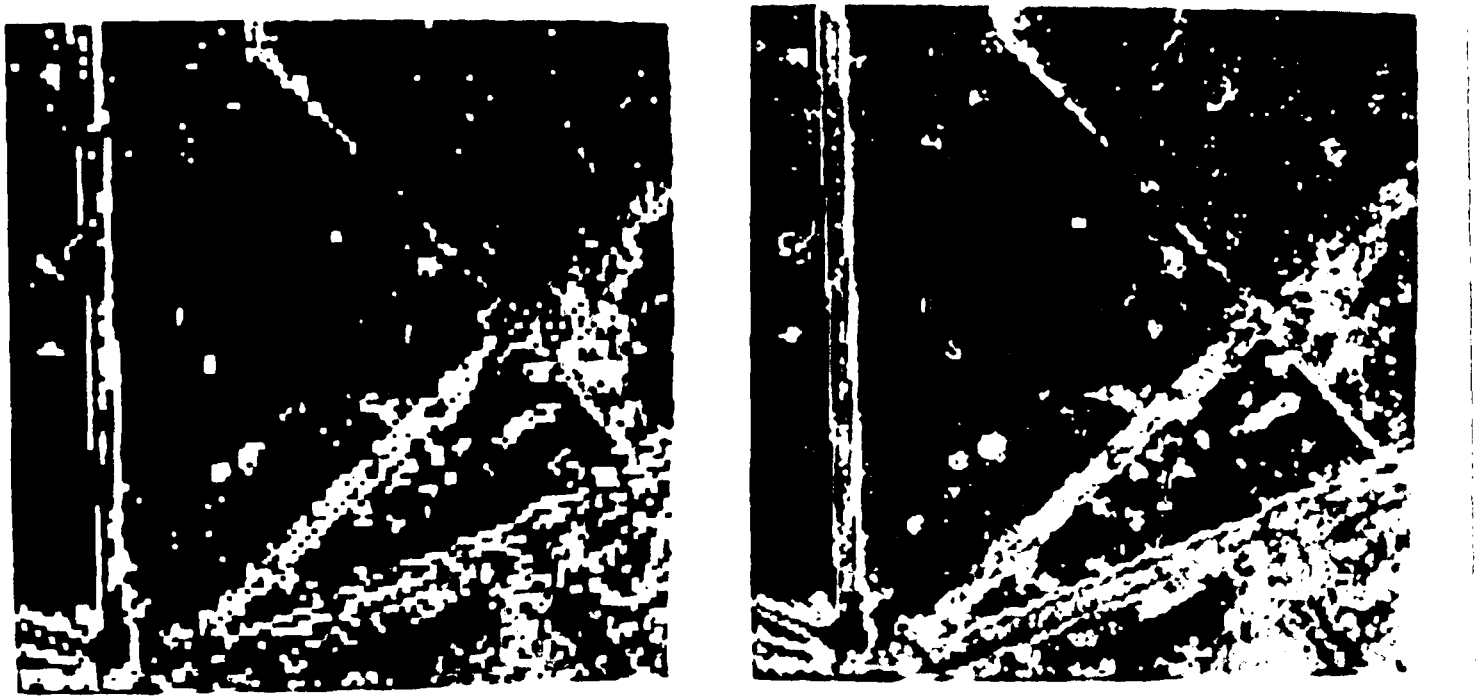
The results showed very little difference when the prior probabilities were changed within each subgroup. Each increase in probability added a few more windows identified as on-road. The difference in performance between the two sizes of windows was not significant except in the first test as seen in Figure 5.6 . The smaller window sizes picked up a large amount of lighter off-road regions as being on-road. Figures 5.7, 5.8 and 5.9 show varying degrees of results with the second and third test group showing the best results.

## D. CONCLUSION

The procedure in running the experiments was to teach the system once for each subgroup and then to run the 7 color scan and the five Gaussian scans. As a result, the exact same off-road regions were used in each of these six scans. This made the data easier to compare.

The results clearly show that the Gaussian type scan produced better results. In three of the four groups selected, the results were very good. The results also indicate that in the teaching portion, the regions chosen can improve the results. In this image, on-road areas are light, with some variance, and off-road regions vary from a few very light areas with no variance to very dark regions with no variance. It is important for the operator to choose either a varying sample of off-road regions as in group 2, or as in group 3, off-road areas that are close but not identical to on-road areas, for the best result.

Figure 5.1 Map Image and 7 Color Scan Codes.



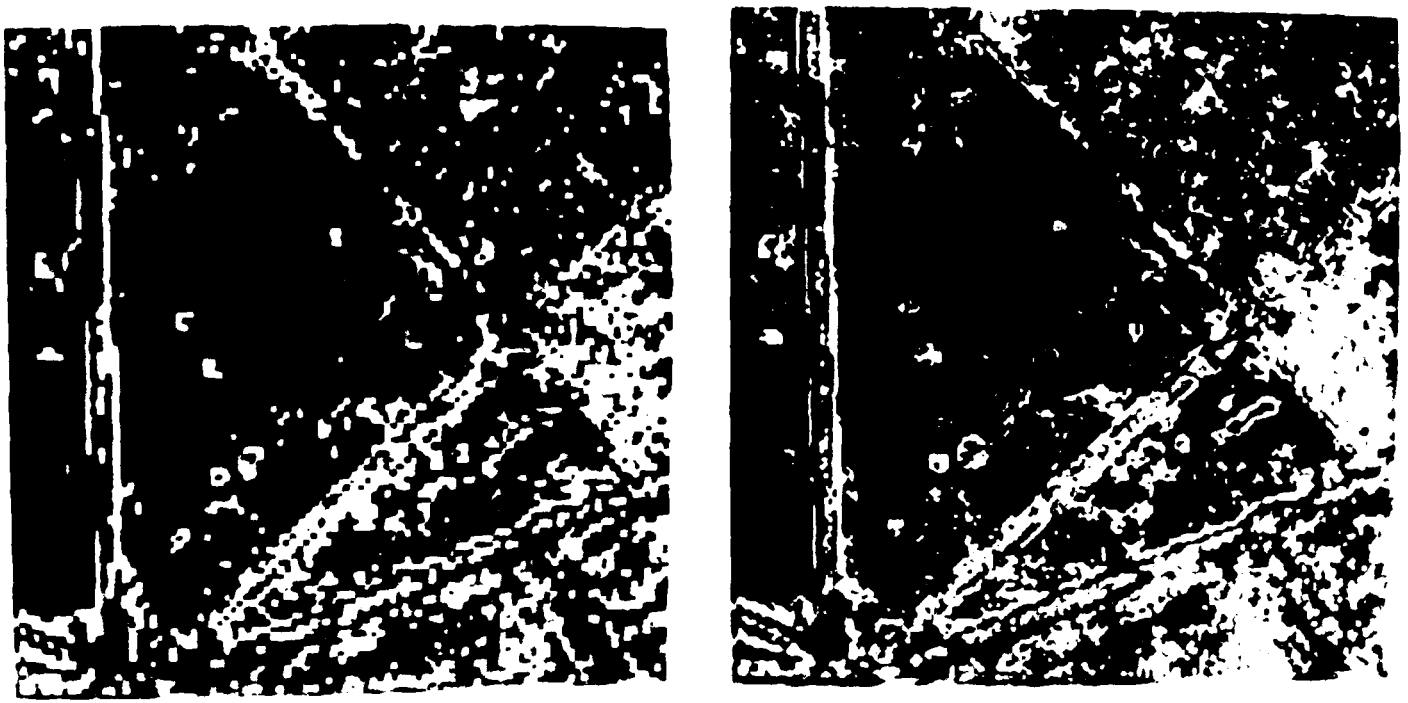
INNER WINDOW SIZE 5 — INNER WINDOW SIZE 3

OUTER WINDOW SIZE 10 — OUTER WINDOW SIZE 9

THESE SCANS USED OFF-ROAD AREAS FROM GROUP I FOR TRAINNING

15 VERY DARK REGIONS

Figure 5.2 - 7 Color Scan, Group 1.



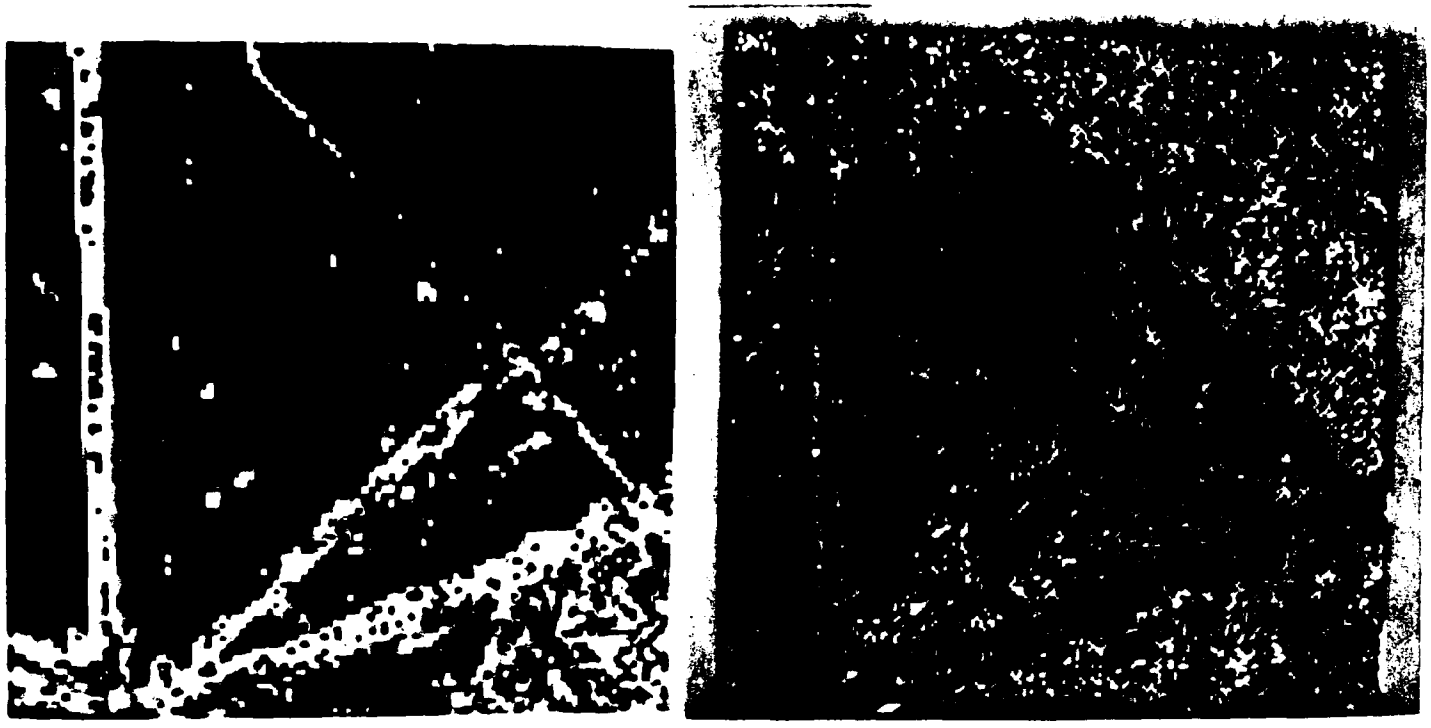
INNER WINDOW SIZE 5
OUTER WINDOW SIZE 10

INNER WINDOW SIZE 3
OUTER WINDOW SIZE 9

THESE SCANS USED OUT-ROAD AREAS FROM GROUP 2 FOR TRAINNING
3 VERY DARK, 4 DARK, 6 LIGHT, 2 VERY LIGHT REGIONS

Figure 5.3 7 Color Scan, Group 2.

Figure 5.4 7 Color Scan, Group 3.



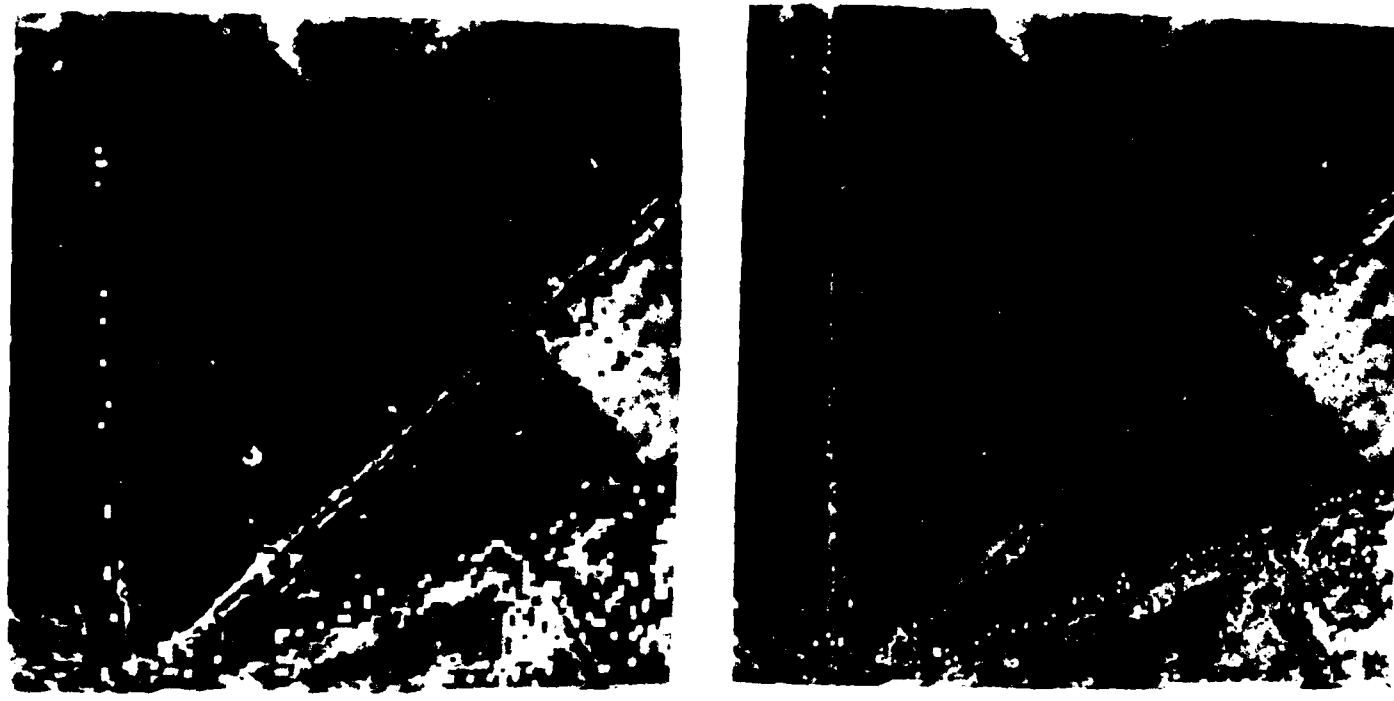
INNER WINDOW SIZE 5
OUTER WINDOW SIZE 10

INNER WINDOW SIZE 3
OUTER WINDOW SIZE 9

THESE SCANS USED OFF-ROAD AREAS FROM GROUP 4 FOR TRAINING
15 VERY TIGHT REGIONS

Figure 5.5 - Color Scan, Group 4

Figure 5.6 Gaussian Scan, Group 1



INNER WINDOW SIZE 5
OUTER WINDOW SIZE 10

INNER WINDOW SIZE 3
OUTER WINDOW SIZE 9

THESE SCANS USED OFF-ROAD AREAS FROM GROUP 2 FOR TRAINNING
3 VERY DARK, 4 DARK, 6 LIGHT, 2 VERY LIGHT REGIONS
PROBABILITY OF AN REGION BEING OFF-ROAD IS 0.5 FOR THESE SCANS

Figure 5.7 Gaussian Scan, Group 2

INNER WINDOW SIZE 5 — INNER WINDOW SIZE 3

OUTER WINDOW SIZE 10 — OUTER WINDOW SIZE 9

THESE SCANS USED OFF-ROAD AREAS FROM GROUP 3 FOR TRAINNING

15 EIGHT REGIONS

PROBABILITY OF AN REGION BEING OFF-ROAD IS 0.5 FOR THESE SCANS

Figure 5.8 Gaussian Scan, Group 3.

Figure 5.9 Gaussian Scan, Group 4.

# VI. SUMMARY AND RECOMMENDATIONS FOR FURTHER RESEARCH

## A. SUMMARY

This study was aimed at the practical problem of locating roads in aerial photographs. Rapid processing and accurate identification of road areas were the main objectives. To achieve these goals, simple texture features were chosen to identify the road areas.

The first approach was to try a binary feature vector. In this approach, each of the three selected features was considered independently. The results of this approach were interesting, but not as good as hoped. The results indicated that the features were informative, but should be evaluated in a different way.

A second approach was chosen. This approach was based on a more complex method of using all three selected features in combination in which the probability of misclassification is minimized. The results of this approach greatly improved the identification of road areas.

Overall, the results obtained are useful in flagging areas for further attention by a human photo interpreter. However, window classification of pixels alone does not adequately separate road from non-road areas. Further processing using syntactic information is needed to obtain better classification of areas.

## B. RECOMMENDATIONS

This work involved simple methods of classifying windows. More work on window classification is needed. A more complex method might be justified if the effectiveness of identifying road areas improved sufficiently. With anticipated improvements to the hardware of the Graphics Laboratory at NPS, it may be possible to use these more complex methods without a substantial increase in the processing time.

The syntactic identification of road areas using an artificial intelligence expert system such as KEE or ART should be investigated. The system illustrated here could be used as a filter to locate possible road areas for the expert system to look at and further classify using syntactical features of road, such as shape and size.

A final recommendation is to investigate the possibility of improving human interaction in the identification of road areas. For example, a human could have available a low resolution aerial photograph of a large area. The computer would have the same area, but at a much higher resolution. This high resolution image could be processed by the computer. The computer could then flag areas for, the human photo expert to consider. The human could then confirm the computers identification using the low resolution photo, or could call up and display the area in question from the high resolution image.

# APPENDIX A
# USER'S MANUAL

## 1. INTRODUCTION

This document describes how to run the software to teach the system the difference between on-road and off-road areas, the software to scan the image to find the road, and the software to display the road areas. It also describes the file format of the digitized image.

## 2. SOFTWARE DOCUMENTATION

### a. File Format

The files containing the digitized map images are created by the EIKONIX camera and associated software on the VAX/VMS system and transferred to the IRIS [Ref. 4: page 15]. This application uses the files for black and white images only. Each file contains an eighty-six byte header followed by the image data. The header is constructed as follows:

1. The first two bytes represent the file type. Format is integer. This will be '1' for all black and white images.
2. The third and forth bytes are the number of lines of image data. Format is integer.
3. The fifth and sixth bytes are the number of columns of image data. Format is integer.
4. The seventh through eighty-six byte are comments. Format is character.

All integer fields are stored with the Least Significant byte(LSB) first. This application assumes all images are 500 columns wide. The image data immediately follows the header and consist of single bytes. Format is byte. The data is stored sequentially by line. For the file format of the scan files on the IRIS, see Table 1 .

### b. Detailed User Instructions

The software is written in Franz LISP and C. It is designed to be run from the IRIS side terminal. The display of the image and color coded on-road areas are directed to the associated IRIS display terminal. In order to run the program, all C functions must be compiled. It is recommended that all LISP functions be complied to increase execution speed.

To run any of the software, it is necessary to be in the LISP environment, and to have the C functions and LISP function loaded. See Figure A.1 for an example of how to do this.

```
(load 'myfilename)
(cfasl "display.o" "_display" 'display_l "c-function" "-lgl")
(cfasl "ginit.o" "_gint" 'gint_l "c-function" "-lgl")
(cfasl "window1.o" "_window1" 'window1_l "c-function" "-lgl")
(cfasl "readsquare.o" "_readsquare" 'readsquare_l "c-function" "-lgl")
(cfasl "writesquare.o" "_writesquare" 'writesquare_l "c-function" "-lgl")
(cfasl "colorcode.o" "_colorcode" 'colorcode_l "c-function" "-lgl")
```

Figure A.1 Sample of Loading Calls.

***1. Map Display***

To display the map image, call the LISP function *init-file*. The parameters are the filename of the file containing the digitized image, the x screen coordinate of the left side of the image, and the y screen coordinate of the top of the image. Example: (*init-file 'map.dig2 0 766*). This will display map.dig2 in the upper right corner of the IRIS display terminal screen.

***2. Instruct Computer***

To teach the computer the difference between on-road and off-road areas on a given image, first display the image as in the example above, insuring that it is displayed at screen coordinates 0,766. Next call the LISP function *main*. Parameters are the inner window size, inner-window-size and number of points to teach with. Example: (*main 9 3 15*). It will then be necessary to use the left mouse button to select the points the computer request. Note the size of the selection box is the size of the outer-window-size.

***3. Scan Map***

To scan the map image, it is necessary to first generate the data produced when the system is taught. This can be accomplished in one of two ways. First of all the scan can be run immediately after the computer is taught, since the required data is assumed to be global. The second choice is to have saved the data from the teaching

phase in a file and retrieve that data. There are two LISP functions *read-data* and ***store-data*** that will accomplish this. Both functions have only the file name as a parameter. Once the data is global, the scanning phase is accomplished by running ***scan-map.*** Example: (*scan-map outer-window-size inner-window-size outfile1 outfile2* ***prob***) Outfile1 is an output file which will contain the results of the seven color scan. Outfile2 is an output file which will contain the results of the gaussian scan. Prob is the probability the scanned window will be an on-road area. Note that the results of the gaussian scan will be displayed in the upper right of the screen. If desired, the image of the map can be displayed there before running the program. This will result in an overlay effect. To display the map use the following: (*init-file filename 510 766*)

***4. Display Road***

To display the road the LISP function *show-road* is used. Parameters are the filename of a file created in *scan-map* and the offset. The road can only be displayed in the upper part of the screen. The offset is the x screen coordinate. Example: (*show-road map.gaussian 510*).

# APPENDIX B
# PROGRAM

```
; This set of functions is designed to identify on-road
; areas in digitized arial photographs
;
;
;
; GLOBAL VARIABLES:
; outer_on_road_mean  : mean of on-road means of outer windows
; inner_on_road_mean  : mean of on-road means of inner windows
; outer_on_road_var   : mean of on-road variances of outer windows
; inner_on_road_var   : mean of on-road variances of inner windows
; outer_off_road_mean : mean of off-road means of outer windows
; inner_off_raod_mean : mean of off-road means of inner windows
; outer_off_road_var  : mean of off-road variances of outer windows
; inner_off_road_var  : mean of off-road variances of inner windows
;
;
; outer_var_on_mean  : variance of on-road means of outer windows
; inner_var_on_mean  : variance of on-road means of inner windows
; outer_var_on_var   : variance of on-road variances of outer windows
; inner_var_on_var   : variance of on-road variances of inner windows
; outer_var_off_mean : variance of off-road means of outer windows
; inner_var_off_mean : variance of off-road means of inner windows
; outer_var_off_var  : variance of off-road variances of outer windows
; inner_var_off_var  : variance of off-road variances of inner windows
;
; var_prod_on_road  : variance product is the ln of the square root
;                     of inner_var_on_mean * outer_var_on_var *
;                        inner_var_on_var
; var_prod_off_road : variance product is the ln of the square root
;                     of inner_var_off_mean * outer_var_off_var *
;                        inner_var_off_var
```

```
; outer_dev_on_mean : standard dev of mean of on-road outer windows
; inner_dev_on_mean : standard dev of mean of on-road inner windows
; outer_dev_on_var  : standard dev of variance of on-road outer windows
; inner_dev_on_var  : standard dev of variance of on-road inner windows
; outer_dev_off_mean: standard dev of mean of off-road outer windows
; inner_dev_off_mean: standard dev of mean of off-road inner windows
; outer_dev_off_var : standard dev of variance of off-road outer windows
; inner_dev_off_var : standard dev of variance of off-road inner windows
;
;
; outer_mean_threshold : threshold for mean of outer windows
; inner_mean_threshold : threshold for mean of inner windows
; outer_var_threshold  : threshold for variance of outer windows
; inner_var_threshold  : threshold for variance of inner windows
```

```
;*************************************************************
; MAIN                                                       *
; Function is to instruct system in the differance between   *
; on-road and off-road areas                                 *
; Inputs:                                                    *
;  outer_window_size: size of outer scan window              *
;  inner_wondow_size: size of inner scan window              *
;  points           : number of points used to teach         *
; Functions:                                                 *
;   window_1                                                 *
;   readsquare_1                                             *
;   a_window_mean                                            *
;   a_window_variance                                        *
;   mean                                                     *
;   variance                                                 *
;   threshold                                                *
;*************************************************************

(defun main (outer_window_size inner_window_size points)
  (setq outer_cursorval (new-vectori-word (expt outer_window_size
   2)))
  (setq inner_cursorval (new-vectori-word (expt inner_window_size
   2)))
  (setq xcursor (new-vectori-word 2))
  (setq ycursor (new-vectori-word 2))

;*
;**** loop to learn on road areas
;*
  (do (( i 0 (+ i 1)))
    ((= i points) t)
;*
;***** read in pixel values for the on road windows
;*
    (window1_1 outer_window_size lines cols xcursor ycursor 1)
    (readsquare_1 (vrefi-word xcursor 0) (vrefi-word ycursor 0)
```

```
                    outer_window_size outer_cursorval)
 (vseti-word xcursor 1(+ (+(-(/ outer_window_size 2)
                                (/ inner_window_size 2))1)
 (vrefi-word xcursor 0)))
 (vseti-word ycursor 1(+ (+(-(/ outer_window_size 2)
                                (/ inner_window_size 2))1)
 (vrefi-word ycursor 0)))
 (readsquare_1 (vrefi-word xcursor 1) (vrefi-word ycursor 1)
                    inner_window_size  inner_cursorval)
(cond ((= i 0)
        (setq outer_on_mean_list (list (a_window_mean
          outer_window_size  outer_cursorval)))
        (setq inner_on_mean_list (list (a_window_mean
          inner_window_size inner_cursorval)))
        (setq outer_on_var_list (list (a_window_variance
                                          outer_window_size
                                          outer_cursorval
                                          (car outer_on_mean_list)
                                         )))
        (setq inner_on_var_list (list (a_window_variance
                                          inner_window_size
                                          inner_cursorval
                                          (car inner_on_mean_list)
                                        ))) )
```

```
          ((> i 0)
           (attach  (a_window_mean outer_window_size
                      outer_cursorval) outer_on_mean_list)
           (attach  (a_window_mean inner_window_size
                      inner_cursorval)inner_on_mean_list)
           (attach   (a_window_variance outer_window_size
                       outer_cursorval (car outer_on_mean_list))
                       outer_on_var_list)
           (attach  (a_window_variance inner_window_size
                      inner_cursorval (car inner_on_mean_list))
                      inner_on_var_list)  )
    )
   )


;*
;***** loop to learn off road areas
;*

    (do (( i 0 (+ i 1)))
     ((= i points) t)

;*
;***** read in pixel values for the off road windows
;*
     (window1_1 outer_window_size lines cols xcursor ycursor 2)
     (readsquare_1 (vrefi-word xcursor 0) (vrefi-word ycursor 0)
                     outer_window_size outer_cursorval)
     (vseti-word xcursor 1(+ (+(-(/ outer_window_size 2)
                                  (/ inner_window_size 2))1)
     (vrefi-word xcursor 0)))
     (vseti-word ycursor 1(+ (+(-(/ outer_window_size 2)
                                  (/ inner_window_size 2))1)
     (vrefi-word ycursor 0)))
     (readsquare_1 (vrefi-word xcursor 1) (vrefi-word ycursor 1)
                     inner_window_size inner_cursorval)
```

```
(cond ((= i 0)
        (setq outer_off_mean_list (list (a_window_mean
          outer_window_size  outer_cursorval)))
        (setq inner_off_mean_list (list (a_window_mean
          inner_window_size inner_cursorval)))
        (setq outer_off_var_list (list  (a_window_variance
                                          outer_window_size
                                          outer_cursorval
                                         (car
                                          outer_off_mean_list)
                                        )))
        (setq inner_off_var_list (list (a_window_variance
                                          inner_window_size
                                          inner_cursorval
                                          (car
                                           inner_off_mean_list)
                                       ))) )
```

```
            ((> i 0)
             (attach  (a_window_mean outer_window_size
                          outer_cursorval) outer_off_mean_list)
             (attach  (a_window_mean inner_window_size
                          inner_cursorval) inner_off_mean_list)
             (attach  (a_window_variance outer_window_size
                          outer_cursorval (car outer_off_mean_list))
                          outer_off_var_list)
             (attach  (a_window_variance inner_window_size
                          inner_cursorval (car inner_off_mean_list))
                          inner_off_var_list)  )
        )
    )

    ;*
    ;***** figure mean of given list
    ;*
    (setq outer_on_road_mean (mean outer_on_mean_list points))
    (setq inner_on_road_mean (mean inner_on_mean_list points))
    (setq outer_on_road_var (mean outer_on_var_list points))
    (setq inner_on_road_var (mean inner_on_var_list points))
    (setq outer_off_road_mean (mean outer_off_mean_list points))
    (setq inner_off_road_mean (mean inner_off_mean_list points))
    (setq outer_off_road_var (mean outer_off_var_list points))
    (setq inner_off_road_var (mean inner_off_var_list points))

    ;*
    ;***** figure variance of given list
    ;*
    (setq outer_var_on_mean (variance outer_on_mean_list points
                                          outer_on_road_mean))
    (setq inner_var_on_mean (variance inner_on_mean_list points
                                          inner_on_road_mean))
    (setq outer_var_on_var (variance outer_on_var_list points
                                          outer_on_road_var))
```

```
(setq inner_var_on_var (variance inner_on_var_list points
                                          inner_on_road_var))
(setq outer_var_off_mean (variance outer_off_mean_list points
                                          outer_off_road_mean))
(setq inner_var_off_mean (variance inner_off_mean_list points
                                          inner_off_road_mean))
(setq outer_var_off_var (variance outer_off_var_list points
                                          outer_off_road_var))
(setq inner_var_off_var (variance inner_off_var_list points
                                           inner_off_road_var))

;*
;***** figure the variance product to be used
;***** in the gaussian function
;*
(setq var_prod_on_road (log (sqrt (float (product inner_var_on_mean
                                    outer_var_on_var inner_var_on_var)))))
(setq var_prod_off_road (log (sqrt (float (product inner_var_off_mean
                                    outer_var_off_var inner_var_off_var)))))
```

```
;*
;***** figure standard dev of given list
;*

(setq outer_dev_on_mean (sqrt (float outer_var_on_mean)))
(setq inner_dev_on_mean (sqrt (float inner_var_on_mean)))
(setq outer_dev_on_var  (sqrt (float outer_var_on_var)))
(setq inner_dev_on_var  (sqrt (float inner_var_on_var)))
(setq outer_dev_off_mean (sqrt (float outer_var_off_mean)))
(setq inner_dev_off_mean (sqrt (float inner_var_off_mean)))
(setq outer_dev_off_var  (sqrt (float outer_var_off_var)))
(setq inner_dev_off_var  (sqrt (float inner_var_off_var)))

;*
;***** figure thresholds for means and vars
;*
(setq outer_mean_threshold (threshold outer_dev_off_mean
                                      outer_dev_on_mean
                                      outer_on_road_mean
                                      outer_off_road_mean))
(setq inner_mean_threshold (threshold inner_dev_off_mean
                                      inner_dev_on_mean
                                      inner_on_road_mean
                                      inner_off_road_mean))
(setq outer_var_threshold (threshold outer_dev_off_var
                                     outer_dev_on_var
                                     outer_on_road_var
                                     outer_off_road_var))
(setq inner_var_threshold (threshold inner_dev_off_var
                                     inner_dev_on_var
                                     inner_on_road_var
                                     inner_off_road_var))
)
```

```
;*************************************************************
; INIT-FILE                                                  *
; Displays image                                             *
; Inputs:                                                    *
;  filename:     name of file holding digitized image        *
;  display-at-x: screen x coordinate (left side of image)    *
;  display-at-y: screen y coordinate (top)                   *
; Functions:                                                 *
;  display_1                                                 *
;*************************************************************

(defun init-file (filename display-at-x display-at-y)
      (display_1 (get_pname filename) display-at-x  display-at-y)
      (setq fd (infile filename))
      (fseek fd 2 0)
      (setq lines (+ (tyi fd) (* 256 (tyi fd))))
      (setq cols  (+ (tyi fd) (* 256 (tyi fd))))
      (close fd)
)
```

```
;*************************************************************
; A_WINDOW_MEAN                                              *
; Finds the mean of a given window                           *
; Inputs:                                                    *
;  windowsize: the size of the window                        *
;  cursorval:  the array holding the values of each pixel    *
;              in the window                                 *
; Returns:                                                   *
;   the mean of the given window                             *
;*************************************************************
(defun a_window_mean (windowsize cursorval)
  (/ (do (( j 1 (+ j 1))
          (sum (vrefi-word cursorval 0)
               (+ sum (vrefi-word cursorval j))))
         ((= j (expt windowsize 2)) sum)
     )(expt windowsize 2))
)

;*************************************************************
; A_WINDOW_VARIANCE                                          *
; find variance in a window                                  *
; Inputs:                                                    *
;  windowsize:  the size of the given window                 *
;  cursorval:   an array holding the value of each pixel     *
;               in the window                                *
;  window_mean: the mean of the given window                 *
; Returns:                                                   *
;  the variance of the given window                          *
;*************************************************************
(defun a_window_variance (windowsize cursorval window_mean)
   (-(/(do ((j 1 (+ j 1))
           (sumsq (expt(vrefi-word cursorval 0) 2)
                  (+ sumsq (expt(vrefi-word cursorval j) 2))))
          ((= j (expt windowsize 2)) sumsq)
       )(expt windowsize 2))(expt window_mean 2))
)
```

```
;*************************************************************
: MEAN                                                        *
; find mean of the input list                                 *
; Inputs:                                                     *
;  list:   a list containing the values you wish to find the  *
;          the mean of                                        *
;  points: the number of points in the list                   *
; Returns:                                                    *
;  the mean of the given list of points                       *
;*************************************************************
(defun mean (list points)
  (/ ( do (( sum (car list) (+ sum (car rest_of_list)))
           (rest_of_list (cdr list) (cdr rest_of_list)))
          (( null rest_of_list) sum) ) points)
)

;*************************************************************
; VARIANCE                                                    *
; find variance  of given list and the mean of                *
; the list, this is not normalized                            *
; Inputs:                                                     *
;   list    : a list of points you wish to find the           *
;             variance of                                     *
;   points  : the number on points in list                    *
;   the_mean: the mean of the list                            *
; Returns;                                                    *
;   the variance of a list of points                          *
;*************************************************************
(defun variance (list points the_mean)
   (quotient (do ((sum (expt (diff (car list) the_mean) 2)
                             (add sum (expt (diff(car rest_of_list)
                                                 the_mean) 2)))
                     (rest_of_list (cdr list) (cdr rest_of_list)))
                    ((null rest_of_list) sum) ) points )
)
```

```
;*************************************************************
; THRESHOLD                                                  *
; find the threshold given two means and two standard dev    *
; Inputs:                                                    *
;   dev_off: standard dev of off road area                   *
;   dev_on : standard dev of on  road area                   *
;   off    : mean of off road area                           *
;   on     : mean of on  road area                           *
; Returns:                                                   *
;  the threshold between the on-road and off-road            *
;*************************************************************

(defun threshold (dev_off dev_on off on)
  (quotient (add (product dev_off on)
                 (product dev_on off))
                 (add dev_on dev_off))
)
```

```
;*************************************************************
; COMPARE                                                    *
;  Given inner or outer, on and off road means or variances  *
;  and the threshold returnes true if on road area           *
; Inputs:                                                    *
;  window    : inner or outer, mean or var, for a window     *
;  threshold : threshold for the given window*
;  true_type : inner or outer, mean or var, for on road*
;  false_type: inner or outer, mean or var, for off road     *
;*************************************************************

(defun compare (window threshold true_type false_type)
  (cond (( or (and (> window threshold)
                   (> true_type false_type)
              )
              (and (< window threshold)
                   (< true_type false_type)
              )
         )
         t
       )
         (t
          ()
         )
   )
)
```

```
;*************************************************************
; GAUSSIAN                                                   *
;  finds the gaussian property of either on-road or off-road *
;  areas.                                                    *
;  Inputs:                                                   *
;    prob              : probality an area is on-road        *
;    var-prod          : variance product                    *
;    window_inner_mean: mean of the given inner window       *
;    window_outer_var  : var  of the given outer window      *
;    window_inner_var  : var  of the giver inner window      *
;    mean_inner_mean   : mean of all the inner windows means *
;    mean_outer_var    : mean of all the outer window vars   *
;    mean_inner_var    : mean of all the inner window vars   *
;    var_inner_mean    : var  of all the inner window means  *
;    var_outer_var     : var  of all the outer window vars   *
;    var_inner_var     : var  of all the inner window vars   *
; Returns:                                                   *
;  the gaussian value                                        *
;*************************************************************
(defun gaussian (prob var_prod
                  window_inner_mean window_outer_var
                  window_inner_var  mean_inner_mean
                  mean_outer_var  mean_inner_var
                  var_inner_mean   var_outer_var   var_inner_var)
  (diff (diff (log prob) var_prod)
     (quotient (add (quotient (expt (diff window_inner_mean
                mean_inner_mean) 2)  var_inner_mean)
                     (quotient (expt (diff window_outer_var
                mean_outer_var ) 2)  var_outer_var )
                     (quotient (expt (diff window_inner_var
                mean_inner_var ) 2)  var_inner_var ) )2))
)
```

```
;*************************************************************
;SCAN-MAP                                                     *
; scans the displayed image, identifies on-road areas and     *
; stores these areas in a file.                               *
; Inputs:                                                     *
;  outer_size     : the size of the outer window              *
;  inner_size     : the size of the inner window              *
;  outfilename1   : file to hold on-road areas from 7 color   *
;                   scan                                      *
;  outfilename2   : file to hold on-road areas from gaussian  *
;                   scan                                      *
;  prob           : probality an area is on-road              *
; Functions:                                                  *
;  readsquare_1                                               *
;  gaussian                                                   *
;  compare                                                    *
;  writesquare_1                                              *
;  store-data                                                 *
;*************************************************************
(defun scan-map (outer_size inner_size outfilename1 outfilename2
                  prob)
  (setq fd1 (outfile outfilename1))
  (setq fd2 (outfile outfilename2))
  (setq on_road_prob prob)
  (store-data  fd1)
  (store-data  fd2)
  (tyo inner_size fd1)
  (tyo inner_size fd2)
  (setq outer_windowvector(new-vectori-word (expt outer_size 2)))
  (setq inner_windowvector(new-vectori-word (expt inner_size 2)))
  (do (( i (+ (- 766 outer_size ) 1) (- i inner_size)))
    ((< i (+ (- 766 lines) 1)) t)
    (setq inner_i (+ i (+ (- (/ outer_size 2)(/ inner_size 2))
                    1)))
    (msg "scan is now scanning map line " i N)
```

```
(do (( j 0 (+ j inner_size)))
  ((> j (- cols outer_size))t)
  (setq inner_j (+ j (+ (- (/ outer_size 2)(/ inner_size 2))
                    1)))
  (readsquare_l j i outer_size outer_windowvector)
  (readsquare_l inner_j inner_i inner_size
                  inner_windowvector)
  (setq outer_window_mean (a_window_mean outer_size
                           outer_windowvector))
  (setq inner_window_mean (a_window_mean inner_size
                           inner_windowvector))
  (setq outer_window_var  (a_window_variance outer_size
                                outer_windowvector
                                outer_window_mean))
  (setq inner_window_var  (a_window_variance inner_size
                                inner_windowvector
                                inner_window_mean))
```

```
;*
;***** the gaussian method for the scan
;*
(cond ((> (gaussian on_road_prob var_prod_on_road
                        inner_window_mean outer_window_var
                        inner_window_var inner_on_road_mean
                        outer_on_road_var inner_on_road_var
                        inner_var_on_mean outer_var_on_var
                        inner_var_on_var)
            (gaussian (diff 1 on_road_prob) var_prod_off_road
                        inner_window_mean outer_window_var
                        inner_window_var  inner_off_road_mean
                        outer_off_road_var inner_off_road_var
                        inner_var_off_mean outer_var_off_var
                        inner_var_off_var)
        )
        (tyo 7 fd2)
        (tyo (car (Divide inner_j 256)) fd2)
        (tyo inner_j fd2)
        (tyo (car (Divide inner_i 256)) fd2)
        (tyo inner_i fd2)
        (writesquare_1 (+ 510 inner_j) inner_i inner_size 255 0 0)
      ) )
```

```
;*
;***** scanning with seven color output
;*
       (cond ( (compare inner_window_mean inner_mean_threshold
                        inner_on_road_mean inner_off_road_mean)
                 (cond ( (compare inner_window_var
                                  inner_var_threshold
                   inner_on_road_var
                   inner_off_road_var)
                          (cond ( (compare outer_window_var
                          outer_var_threshold
                          outer_on_road_var
                          outer_off_road_var)
  ;(writesquare_1 (+ 510 inner_j) inner_i inner_size 255 0 0)
                                    (tyo 7 fd1)
                                    (tyo (car (Divide inner_j 256))
                                     fd1)
                                    (tyo inner_j fd1)
                                    (tyo (car (Divide inner_i 256))
                                     fd1)
                                    (tyo inner_i fd1)
                                  )
                                  ( t
;(writesquare_1 (+ 510 inner_j) inner_i inner_size 0 255 255)
                                    (tyo 4 fd1)
                                    (tyo (car (Divide inner_j 256))
                                     fd1)
                                    (tyo inner_j fd1)
                                    (tyo (car (Divide inner_i 256))
                                     fd1)
                                    (tyo inner_i fd1)
                                  )
                            )
                          )
                          ( (cond ( (compare outer_window_var
```

```
                                        outer_var_threshold
                                        outer_on_road_var
                                        outer_off_road_var)
;(writesquare_1 (+ 510 inner_j) inner_i inner_size 150 100 100)
                                    (tyo 6 fd1)
                                    (tyo (car (Divide inner_j 256))
                                     fd1)
                                    (tyo inner_j fd1)
                                    (tyo (car (Divide inner_i 256))
                                     fd1)
                                    (tyo inner_i fd1)
                                  )
                                  ( t
;(writesquare_1 (+ 510 inner_j) inner_i inner_size 0 255 0)
                                    (tyo 1 fd1)
                                    (tyo (car (Divide inner_j 256))
                                     fd1)
                                    (tyo inner_j fd1)
                                    (tyo (car (Divide inner_i 256))
                                     fd1)
                                    (tyo inner_i fd1)
                                  )
                              )
                            )
                          )
                        )
                        ( (cond ( (compare inner_window_var
                                           inner_var_threshold
                                           inner_on_road_var
                                           inner_off_road_var)
                                  (cond ( (compare outer_window_var
                                                   outer_var_threshold
                                                   outer_on_road_var
                                                   outer_off_road_var)
;(writesquare_1 (+ 510 inner_j) inner_i inner_size 255 0 255)
```

```
                                    (tyo 5 fd1)
                                    (tyo (car (Divide inner_j 256))
                                        fd1)
                                    (tyo inner_j fd1)
                                    (tyo (car (Divide inner_i 256))
                                        fd1)
                                    (tyo inner_i fd1)
                                  )
                                  ( t
;(writesquare_1 (+ 510 inner_j) inner_i inner_size 0 0 255)
                                    (tyo 2 fd1)
                                    (tyo (car (Divide inner_j 256))
                                      fd1)
                                    (tyo inner_j fd1)
                                    (tyo (car (Divide inner_i 256))
                                     fd1)
                                    (tyo inner_i fd1)
                                  )
                              )
                          )
                          ( (compare outer_window_var
                                     outer_var_threshold
                                     outer_on_road_var
                                     outer_off_road_var)
;(writesquare_1 (+ 510 inner_j) inner_i inner_size 255 255 0)
                            (tyo 3 fd1)
                            (tyo (car (Divide inner_j 256)) fd1)
                            (tyo inner_j fd1)
                            (tyo (car (Divide inner_i 256)) fd1)
                            (tyo inner_i fd1)
                          )
                      )
                  )   ) ) )  (close fd1) (close fd2)  )
```

```
;*************************************************************
; STORE-DATA                                                 *
; fixes the data up so it can be placed in a file.           *
; This version of LISP only stores fixnumbers                *
; Inputs:                                                    *
;  fd : filename of file to store data in                    *
;*************************************************************

(defun store-data (fd)
  (tyo (car (Divide (fix outer_on_road_mean) 256)) fd)
  (tyo (fix outer_on_road_mean) fd)
  (tyo (car (Divide (fix inner_on_road_mean) 256)) fd)
  (tyo (fix inner_on_road_mean) fd)

  (tyo (fix (quotient (product outer_dev_on_mean  1000) 65536)) fd )
  (tyo  (fix (quotient
              [diff (product outer_dev_on_mean 1000) (product
              (fix (quotient (product outer_dev_on_mean 1000)
               65536)) 65536)- 256)) fd)
  (tyo  (fix (product outer_dev_on_mean 1000)) fd)

  (tyo  (fix (quotient (product inner_dev_on_mean  1000) 65536))
  fd)
  (tyo  (fix (quotient
              [diff (product inner_dev_on_mean 1000) (product
              (fix (quotient (product inner_dev_on_mean 1000)
                                  65536)) 65536)-  256)) fd)
  (tyo  (fix (product inner_dev_on_mean 1000)) fd)

  (tyo (car (Divide (fix outer_on_road_var) 256)) fd)
  (tyo (fix outer_on_road_var) fd)

  (tyo (car (Divide (fix inner_on_road_var) 256)) fd)
  (tyo (fix inner_on_road_var) fd)

  (tyo (fix (quotient (product outer_dev_on_var 1000) 65536)) fd)
  (tyo  (fix (quotient
              [diff (product outer_dev_on_var 1000) (product
```

```
                (fix (quotient (product outer_dev_on_var 1000)
                                        65536)) 65536)- 256)) fd)
(tyo  (fix (product outer_dev_on_var 1000)) fd)

(tyo  (fix (quotient (product inner_dev_on_var 1000) 65536)) fd)
(tyo  (fix (quotient
                [diff (product inner_dev_on_var 1000) (product
                (fix (quotient (product inner_dev_on_var 1000)
                                        65536)) 65536)- 256)) fd)
(tyo  (fix (product inner_dev_on_var 1000)) fd)

(tyo (car (Divide (fix outer_off_road_mean) 256)) fd)
(tyo (fix outer_off_road_mean) fd)
(tyo (car (Divide (fix inner_off_road_mean) 256)) fd)
(tyo (fix inner_off_road_mean) fd)

(tyo (fix (quotient (product outer_dev_off_mean  1000) 65536)) fd)
(tyo  (fix (quotient
                [diff (product outer_dev_off_mean 1000) (product
                (fix (quotient (product outer_dev_off_mean 1000)
                                        65536)) 65536)- 256)) fd)
(tyo  (fix (product outer_dev_off_mean 1000)) fd)

(tyo (fix (quotient (product inner_dev_off_mean  1000) 65536)) fd)
(tyo  (fix (quotient
                [diff (product inner_dev_off_mean 1000) (product
                (fix (quotient (product inner_dev_off_mean 1000)
                                        65536)) 65536)- 256)) fd)
(tyo  (fix (product inner_dev_off_mean 1000)) fd)

(tyo (car (Divide (fix outer_off_road_var) 256)) fd)
(tyo (fix outer_off_road_var) fd)

(tyo (car (Divide (fix inner_off_road_var) 256)) fd)
(tyo (fix inner_off_road_var) fd)

(tyo (fix (quotient (product outer_dev_off_var 1000) 65536)) fd)
(tyo  (fix (quotient
                [diff (product outer_dev_off_var 1000) (product
```

```
                (fix (quotient (product outer_dev_off_var 1000)
                                        65536)) 65536)- 256)) fd)
(tyo  (fix (product outer_dev_off_var 1000)) fd)

(tyo (fix (quotient (product inner_dev_off_var 1000) 65536)) fd)
(tyo  (fix (quotient
                [diff (product inner_dev_off_var 1000) (product
                (fix (quotient (product inner_dev_off_var 1000)
                                        65536)) 65536)- 256)) fd)
(tyo  (fix (product inner_dev_off_var 1000)) fd)

(tyo  (fix (quotient (product outer_mean_threshold 1000)
                                                65536)) fd)
(tyo  (fix (quotient
                [diff (product outer_mean_threshold 1000) (product
                (fix (quotient (product outer_mean_threshold
                                1000)65536)) 65536)- 256)) fd)
(tyo  (fix (product outer_mean_threshold 1000)) fd)

(tyo  (fix (quotient (product inner_mean_threshold 1000)
                65536)) fd)
(tyo  (fix (quotient
                [diff (product inner_mean_threshold 1000) (product
                (fix (quotient (product inner_mean_threshold
                                1000)65536)) 65536)- 256)) fd)
(tyo  (fix (product inner_mean_threshold 1000)) fd)

(tyo (fix (quotient (product outer_var_threshold 1000) 65536)) fd)
(tyo  (fix (quotient
                [diff (product outer_var_threshold 1000) (product
                (fix (quotient (product outer_var_threshold
                        1000)65536)) 65536)] 256)) fd)
(tyo  (fix (product outer_var_threshold 1000)) fd)

(tyo (fix (quotient (product inner_var_threshold 1000) 65536)) fd)
(tyo  (fix (quotient
                [diff (product inner_var_threshold 1000) (product
                 (fix (quotient (product inner_var_threshold
```

```
                                    1000)65536)) 65536)- 256)) fd)
(tyo  (fix (product inner_var_threshold 1000)) fd)

(tyo (fix (quotient (product var_prod_on_road 1000) 65536)) fd)
(tyo  (fix (quotient
              [diff (product var_prod_on_road 1000) (product
              (fix (quotient (product var_prod_on_road 1000)
                                        65536)) 65536)- 256)) fd)
(tyo  (fix (product var_prod_on_road 1000)) fd)

(tyo (fix (quotient (product var_prod_off_road 1000) 65536)) fd)
(tyo  (fix (quotient
              [diff (product var_prod_off_road 1000) (product
              (fix (quotient (product var_prod_off_road 1000)
                                      65536)) 65536)] 256)) fd)
(tyo  (fix (product var_prod_off_road 1000)) fd)

(tyo  (fix (quotient (product on_road_prob 1000) 65536)) fd)
(tyo  (fix (quotient
              [diff (product on_road_prob 1000) (product
              (fix (quotient (product on_road_prob 1000) 65536))
                                                  65536)]  256)) fd)
(tyo  (fix (product on_road_prob 1000)) fd)

(tyo  (fix (quotient (product (diff 1 on_road_prob) 1000)
                                              65536)) fd)

(tyo  (fix (quotient
              [diff (product (diff 1 on_road_prob) 1000)
           (product (fix (quotient (product (diff 1
        on_road_prob)1000)65536)) 65536)] 256)) fd)
(tyo (fix (product (diff 1 on_road_prob) 1000)) fd)

(tyo (fix (quotient (product inner_on_road_mean 1000) 65536)) fd)
(tyo  (fix (quotient
              [diff (product inner_on_road_mean 1000) (product
              (fix (quotient (product inner_on_road_mean 1000)
                 65536)) 65536)]  256)) fd)
```

```
(tyo  (fix (product inner_on_road_mean 1000)) fd)

(tyo  (fix (quotient  inner_var_on_mean 65536)) fd)
(tyo  (fix (quotient
            (diff  inner_var_on_mean (product
            (fix (quotient inner_var_on_mean 65536)) 65536))
            256)) fd)
(tyo  (fix  inner_var_on_mean) fd)

(tyo  (fix (quotient  outer_var_on_var  65536)) fd)
(tyo  (fix (quotient
            (diff  outer_var_on_var  (product
            (fix (quotient outer_var_on_var  65536)) 65536))
           256)) fd)
(tyo  (fix  outer_var_on_var ) fd)

(tyo (fix (quotient (product inner_var_on_var 1000) 65536)) fd)
(tyo  (fix (quotient
            [diff (product inner_var_on_var 1000) (product
            (fix (quotient (product inner_var_on_var 1000)
                           65536)) 65536)] 256)) fd)
(tyo  (fix (product inner_var_on_var 1000)) fd)

(tyo (fix (quotient (product inner_off_road_mean 1000) 65536)) fd)
(tyo  (fix (quotient
            (diff (product inner_off_road_mean 1000) (product
             (fix (quotient (product inner_off_road_mean 1000)
                                    65536)) 65536)) 256)) fd)
(tyo  (fix (product inner_off_road_mean 1000)) fd)

(tyo  (fix (quotient  inner_var_off_mean 65536)) fd)
(tyo  (fix (quotient
            (diff  inner_var_off_mean (product
            (fix (quotient inner_var_off_mean 65536)) 65536))
           256)) fd)
(tyo  (fix  inner_var_off_mean) fd)

(tyo  (fix (quotient (product outer_var_off_var 1000) 65536))
```

```
 fd)
  (tyo  (fix (quotient
                (diff (product outer_var_off_var 1000) (product
                (fix (quotient (product outer_var_off_var 1000)
                                        65536)) 65536)) 256)) fd)
  (tyo  (fix (product outer_var_off_var 1000)) fd)

  (tyo (fix (quotient (product inner_var_off_var 1000) 65536)) fd)
  (tyo  (fix (quotient
                [diff (product inner_var_off_var 1000) (product
                (fix (quotient (product inner_var_off_var 1000)
               65536)) 65536)]  256)) fd)
  (tyo  (fix (product inner_var_off_var 1000)) fd)

)
```

```
;*************************************************************
; READ-DATA                                                  *
;  reads data from a file                                    *
; Inputs:                                                    *
;  fd : filename to read data in from                        *
;*************************************************************
(defun read-data (fd)
(setq outer_on_road_mean  (+  (* 256 (tyi fd)) (tyi fd)))
(setq inner_on_road_mean  (+  (* 256 (tyi fd)) (tyi fd)))
(setq outer_dev_on_mean (quotient (float (+ (* 65536 (tyi fd ))(*
                                    256 (tyi fd )) (tyi fd))) 1000))
(setq inner_dev_on_mean (quotient (float (+ (* 65536 (tyi fd ))(*
                                    256 (tyi fd )) (tyi fd))) 1000))

(setq outer_on_road_var  (+  (* 256 (tyi fd)) (tyi fd)))
(setq inner_on_road_var  (+  (* 256 (tyi fd)) (tyi fd)))
(setq outer_dev_on_var (quotient (float (+ (* 65536 (tyi fd ))(*
                                  256 (tyi fd ))  (tyi fd))) 1000))
(setq inner_dev_on_var (quotient (float (+ (* 65536 (tyi fd ))(*
                                   256 (tyi fd )) (tyi fd))) 1000))


(setq outer_off_road_mean  (+  (* 256 (tyi fd)) (tyi fd)))
(setq inner_off_road_mean  (+  (* 256 (tyi fd)) (tyi fd)))
(setq outer_dev_off_mean (quotient (float (+ (* 65536 (tyi fd
                        ))(* 256 (tyi fd)) (tyi fd))) 1000))
(setq inner_dev_off_mean (quotient (float (+ (* 65536 (tyi fd
                        ))(* 256 (tyi fd)) (tyi fd))) 1000))

(setq outer_off_road_var  (+  (* 256 (tyi fd)) (tyi fd)))
(setq inner_off_road_var  (+  (* 256 (tyi fd)) (tyi fd)))
(setq outer_dev_off_var (quotient (float (+ (* 65536 (tyi fd ))(*
                                    256 (tyi fd )) (tyi fd))) 1000))
(setq inner_dev_off_var (quotient (float (+ (* 65536 (tyi fd ))(*
                                    256 (tyi fd )) (tyi fd))) 1000))


(setq outer_mean_threshold (quotient (float(+(* 65536 (tyi fd))(*
```

```
                                        256 (tyi fd )) (tyi fd))) 1000))
(setq inner_mean_threshold (quotient (float(+(* 65536 (tyi fd))(*
                                        256 (tyi fd )) (tyi fd))) 1000))
(setq outer_var_threshold (quotient (float(+(* 65536 (tyi fd))(*
                                       256 (tyi fd )) (tyi fd))) 1000))
(setq inner_var_threshold (quotient (float(+(* 65536 (tyi fd))(*
                                       256 (tyi fd )) (tyi fd))) 1000))

(setq var_prod_on_road (quotient (float(+(* 65536 (tyi fd))(* 256
                                         (tyi fd )) tyi fd))) 1000))
(setq var_prod_off_road (quotient (float(+(* 65536 (tyi fd))(*
                                    256 (tyi fd ))(tyi fd))) 1000))
(setq on_road_prob (quotient (float(+(* 65536 (tyi fd))(* 256
                                (tyi fd )) (tyi fd))) 1000))
(setq off_road_prob (quotient (float(+(* 65536 (tyi fd))(* 256
                                  (tyi fd )) (tyi fd))) 1000))

(setq inner_on_road_mean (quotient (float(+(* 65536 (tyi fd))(*
                                    256 (tyi fd )) (tyi fd))) 1000))
(setq inner_var_on_mean   (float(+(* 65536 (tyi fd))(* 256 (tyi
                                                    fd )) (tyi fd))) )
(setq outer_var_on_var (float(+(* 65536 (tyi fd))(* 256 (tyi fd
                                                      )) (tyi fd))) )
(setq inner_var_on_var (quotient (float(+(* 65536 (tyi fd))(* 256
                                       (tyi fd )) (tyi fd))) 1000))

(setq inner_off_road_mean (quotient (float(+(* 65536 (tyi fd))(*
                                    256 (tyi fd )) (tyi fd))) 1000))
(setq inner_var_off_mean   (float(+(* 65536 (tyi fd))(* 256 (tyi
                                    fd )) (tyi fd))) )
(setq outer_var_off_var (quotient (float(+(* 65536 (tyi fd))(*
                                256 (tyi fd )) (tyi fd))) 1000))
(setq inner_var_off_var (quotient (float(+(* 65536 (tyi fd))(*
                                  256 (tyi fd )) (tyi fd))) 1000))
)
```

```
;*************************************************************
; SHOW-ROAD                                                  *
;  display road already found                                *
;  Inputs:                                                   *
;   filename: file where road is stored                      *
;   offset  : x screen coordinate when image starts          *
;  Functions:                                                *
;   read-data                                                *
;   colorcode_1                                              *
;*************************************************************

(defun show-road (filename offset)
  (setq fd2 (infile filename))
  (read-data fd2)
  (setq size (tyi fd2))
  (colorcode_1 size)
  (do ()
     ((= (tyipeek fd2) -1))
     (setq color (tyi fd2))
     (setq x  (+ (+ (* (tyi fd2) 256)(tyi fd2)) offset) )
     (setq y  (+ (* (tyi fd2) 256)(tyi fd2)) )
       (cond ( (= color 1) (writesquare_1 x y size 0 255 0))
             ( (= color 2) (writesquare_1 x y size 0 0 255))
             ( (= color 3) (writesquare_1 x y size 255 255 0))
             ( (= color 4) (writesquare_1 x y size 0 255 255))
             ( (= color 5) (writesquare_1 x y size 255 0 255))
             ( (= color 6) (writesquare_1 x y size 150 100 100))
             ( (= color 7) (writesquare_1 x y size 255 0 0))
       )
  )
  (close fd2)
)
```

```
;*************************************************************
; PRINTMSG                                                   *
;  prints all values on screen                               *
;*************************************************************

(defun printmsg ()
(msg " outer_on_road_mean   " outer_on_road_mean N)
(msg " inner_on_road_mean   " inner_on_road_mean N)
(msg " outer_dev_on_mean    " outer_dev_on_mean  N)
(msg " inner_dev_on_mean    " inner_dev_on_mean  N)

(msg " outer_on_road_var    " outer_on_road_var  N)
(msg " inner_on_road_var    " inner_on_road_var  N)
(msg " outer_dev_on_var     " outer_dev_on_var   N)
(msg " inner_dev_on_var     " inner_dev_on_var   N)

(msg " outer_off_road_mean  " outer_off_road_mean N)
(msg " inner_off_road_mean  " inner_off_road_mean N)
(msg " outer_dev_off_mean   " outer_dev_off_mean  N)
(msg " inner_dev_off_mean   " inner_dev_off_mean  N)

(msg " outer_off_road_var   " outer_off_road_var  N)
(msg " inner_off_road_var   " inner_off_road_var  N)
(msg " outer_dev_off_var    " outer_dev_off_var   N)
(msg " inner_dev_off_var    " inner_dev_off_var   N)


(msg " outer_mean_threshold " outer_mean_threshold N)
(msg " inner_mean_threshold " inner_mean_threshold N)
(msg " outer_var_threshold  " outer_var_threshold  N)
(msg " inner_var_threshold  " inner_var_threshold  N)


(msg " var_prod_on_road     " var_prod_on_road     N)
(msg " var_prod_off_road    " var_prod_off_road    N)
(msg " on_road_prob         " on_road_prob         N)
(msg " off_road_prob        " off_road_prob        N)
```

```
(msg " inner_on_road_mean    " inner_on_road_mean    N)
(msg " inner_var_on_mean     " inner_var_on_mean     N)
(msg " outer_var_on_var      " outer_var_on_var      N)
(msg " inner_var_on_var      " inner_var_on_var      N)
(msg " inner_off_road_mean   " inner_off_road_mean   N)
(msg " inner_var_off_mean    " inner_var_off_mean    N)
(msg " outer_var_off_var     " outer_var_off_var     N)
(msg " inner_var_off_var     " inner_var_off_var     N)
)
```

```
/*********************************************************************/
/* this is an IRIS-2400 program */
/* this is file digit.c
   Its purpose is to read throught a digitizer file
   and produce an image on the IRIS...
*/
#include "gl.h"
#include "device.h"
display(filename)
char filename[26];

   int fd;   /* file descriptor */
   static RGBvalue buffer1[4096];
   long i;   /* temp loop counter */
   short lines, cols, type;
   short mask = 255;
   char comment[80];
   char  byt1[2], byt2[2], byt3[2];
   /* init the graphics system */
   ginit();
   /* put into rgb mode */
   RGBmode();
   /* configure the iris...*/
   gconfig();
   /* set the color to cyan */
   RGBcolor(0,255,255);
   /* draw a cyan background */
   rectfi(0,0,1023,768);
   /* open the file... */
    fd = open(filename,0);
   if(fd < 0)
```

```
    printf("problem in C file indictes not open! n");
    exit(1);

/* read the header information off the file */
read(fd,byt1,2);
read(fd,byt2,2);
read(fd,byt3,2);
read(fd,comment,80);
type = byt1[1] << 8;
type = type  (mask & byt1[0]);
lines = byt2[1] << 8;
lines = lines  (mask & byt2[0]);
cols = byt3[1] << 8;
cols = cols  (mask & byt3[0]);

          /* read through the rows of the file... */
          for(i=0; i < lines; i=i+1)

               /* read a row from the file */
               read(fd,buffer1,cols);
               /* draw that line */
               cmov2i(0,(766-i));
               writeRGB(cols,buffer1,buffer1,buffer1);

close(fd);   /* close the file */
```

```
/************************************************************/
/* window1 This is an IRIS 2400 program
    this sets up the cursor and allows the mouse button
    to be used to pick off points,  it returens the lower
    left conor of the cursor.
/*
#include "gl.h"
#include "device.h"
#include <stdio.h>
window1(size, xmax, ymax, xcursorpos, ycursorpos, type)
int size, xmax, ymax, type;
short int xcursorpos[], ycursorpos[];

static unsigned short cursordef1[16]=
0xf800,0xf800,0xd800,0xf800 ,
0xf800,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000  ;
static unsigned short cursordef2[16]=
0xfc00,0xfc00,0xcc00,0xcc00 ,
0xfc00,0xfc00,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000  ;
static unsigned short cursordef3[16]=
0xfe00,0xfe00,0xc600,0xc600 ,
0xc600,0xfe00,0xfe00,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000  ;
static unsigned short cursordef4[16]=
0xff00,0xff00,0xc300,0xc300 ,
0xc300,0xc300,0xff00,0xff00,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000  ;
static unsigned short cursordef5[16]=
0xff80,0xff80,0xc180,0xc180 ,
```

```
0xc180,0xc180,0xc180,0xff80,
0xff80,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000  ;
static unsigned short cursordef6[16]=
0xffc0,0xffc0,0xc0c0,0xc0c0 ,
0xc0c0,0xc0c0,0xc0c0,0xc0c0,
0xffc0,0xffc0,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000  ;
static unsigned short cursordef7[16]=
0xffe0,0xffe0,0xc060,0xc060 ,
0xc060,0xc060,0xc060,0xc060,
0xc060,0xffe0,0xffe0,0x0000,
0x0000,0x0000,0x0000,0x0000  ;
static unsigned short cursordef8[16]=
0xfff0,0xfff0,0xc030,0xc030 ,
0xc030,0xc030,0xc030,0xc030,
0xc030,0xc030,0xfff0,0xfff0,
0x0000,0x0000,0x0000,0x0000  ;
static unsigned short cursordef9[16]=
0xfff8,0xfff8,0xc018,0xc018 ,
0xc018,0xc018,0xc018,0xc018,
0xc018,0xc018,0xc018,0xfff8,
0xfff8,0x0000,0x0000,0x0000  ;
static unsigned short cursordef10[16]=
0xfffc,0xfffc,0xc00c,0xc00c ,
0xc00c,0xc00c,0xc00c,0xc00c,
0xc00c,0xc00c,0xc00c,0xc00c,
0xfffc,0xfffc,0x0000,0x0000  ;
static unsigned short cursordef11[16]=
0xfffe,0xfffe,0xc006,0xc006 ,
0xc006,0xc006,0xc006,0xc006,
0xc006,0xc006,0xc006,0xc006,
0xc006,0xfffe,0xfffe,0x0000  ;
static unsigned short cursordef12[16]=
0xffff,0xffff,0xc003,0xc003 ,
```

```
0xc003,0xc003,0xc003,0xc003,
0xc003,0xc003,0xc003,0xc003,
0xc003,0xc003,0xffff,0xffff  ;
/* value returns form the event queue */
short value,valuex,valuey;
/* device name returned from the event queue */
Device temp;

RGBmode();
RGBwritemask(0xffff,0xffff,0xffff);
setvaluator(MOUSEX,xmax,0,xmax);
setvaluator(MOUSEY,ymax,766-ymax,766);

defcursor(1 , cursordef1 );
defcursor(2 , cursordef2 );
defcursor(3 , cursordef3 );
defcursor(4 , cursordef4 );
defcursor(5 , cursordef5 );
defcursor(6 , cursordef6 );
defcursor(7 , cursordef7 );
defcursor(8 , cursordef8 );
defcursor(9 , cursordef9 );
defcursor(10, cursordef10);
defcursor(11, cursordef11);
defcursor(12, cursordef12);
/* make button gits on the mouse go to the event queue */
qdevice(MOUSE1);
/* if button 1 is hit, report mousex and y */
tie(MOUSE1,MOUSEX,MOUSEY);
attachcursor(MOUSEX,MOUSEY);
curson();
curorigin(size,2,2);
RGBcursor(size,255,0,0,0xffff,0xffff,0xffff);
```

```
RGBcolor (0,255,255);
rectfi(550,400, 1023,768);
RGBcolor(0,0,0);
cmov2i(550,500);
while (getbutton(MOUSE1) != 0);
if(type == 1)
  charstr("choose onroad position");
  else charstr("                    choose offroad position");
qreset();
while(qtest() == 0);
while(qtest() != 0 )

/*
 printf(" got inside the while  n");
*/
 if(qread(&value) == MOUSE1)

  cursoff();
  temp = qread(&valuex);
  temp = qread(&valuey);
  RGBcolor (0,255,255);
  rectfi(550,400, 1023,768);
  xcursorpos[0] = valuex ;
  ycursorpos[0] = valuey ;
   /* clean-up  */
  unqdevice(MOUSE1);
  qreset();
  return(type);
```

```
/*************************************************************/
/* Readsquare
    reads a square from the screen withe lover left screen
    coordinates of x,y . A square of size size, and returns
    the red bit plane value in the array rbyte
*/

#include "gl.h"
#include "device.h"
#include <stdio.h>
readsquare(x, y, size, rbyte)
int x, y, size;
short int rbyte[];

  int i,j;  /* x,y */
  RGBvalue r[4096], g[4096], b[4096];
  RGBmode();
  for (j = 0 ; j < size; j++)

    cmov2i(x, y+j);
    readRGB(size, r, g, b);
    for (i = 0; i < size; i++)
      rbyte[j * size + i] = r[i];
```

```
/*************************************************************/
/* Writesquare
    writes a square of size size, at screen coordinates
    x, y. In the mapcolor found with red, green, and blue
*/

writesquare(x, y, size, red, green, blue)
int x, y, size, red, green, blue;

  RGBmode();
  RGBcolor(red, green, blue);
  rectfi(x,y, x+size, y+size);
```

```
/*************************************************************/
/* Colorcode
    Prints on the screen the color codes used for the seven
    color scan.
*/
colorcode(size)
int size;

  RGBmode();
  RGBcolor(0, 0, 0);
  rectfi(5, 90, 400, 240);
  RGBcolor(0, 255, 0);
  rectfi(20, 220, 20+size-1, 220+size-1);
  cmov2i(40, 220);
  charstr("mean");
  RGBcolor(0, 0, 255);
  rectfi(20, 200, 20+size-1, 200+size-1);
  cmov2i(40, 200);
  charstr("inner variance");
  RGBcolor(255, 255, 0);
  rectfi(20, 180, 20+size-1, 180+size-1);
  cmov2i(40, 180);
  charstr("outer variance");
  RGBcolor(0, 255, 255);
  rectfi(20, 160, 20+size-1, 160+size-1);
  cmov2i(40, 160);
  charstr("mean and inner variance");
  RGBcolor(255, 0, 255);
  rectfi(20, 140, 20+size-1, 140+size-1);
  cmov2i(40, 140);
  charstr("both variance");
  RGBcolor(150, 100, 100);
  rectfi(20, 120, 20+size-1, 120+size-1);
```

```
cmov2i(40, 120);
charstr("mean and outer variance");
RGBcolor(255, 0, 0);
rectfi(20, 100, 20+size-1, 100+size-1);
cmov2i(40, 100);
charstr("mean and both variance");
```

# LIST OF REFERENCES


1. Burt, P.J., "Fast Algorithms for Estimating Local Image Properties," *Computer Vision, Graphics, and Image Processing*, v. 21, No. 3, March 1983.

2. Conners, R.W., Trivedi, M.M., and Harlow, C.A., "Segmentation of High-Resolution Urban Scene Using Texture Operators," *Computer Vision, Graphics, and Image Processing*, v. 25, No. 3, March 1984.

3. Van Gool, L., Dewaele, P. and Oosterlinck, A., "Survey Texture Analysis Anno 1983," *Computer Vision, Graphics, and Image Processing*, v. 29 No. 3, March 1985.

4. Naval Postgraduate School Report NPS52-87-038, *Using the EIKONIX Digitizer Camera with the IRIS Graphics Workstation*, by J.M. Sando, T.S. Wetherald, and M.J. Zyda, August 1987.

5. Charniak, E. and McDermott, D., *Introduction to Artificial Intelligence*, Addison-Wesley Publishing Company, 1985.

6. Treisman, A., "Preattentive Processing in Vision," *Computer Vision, Graphics, and Image Processing*, v. 31, No. 2, August 1985.

7. Levine, M.D., *Vision in Man and Machine*, McGraw-Hill Book Company, 1985.

8. Fu, K.S., Gonzalez, R.C., and Lee, C.S.G., *Robotics: Control, Sensing, Vision and Intelligence*, McGraw-Hill Book Co., 1987.

9. Parzen, E., *Modern Probability Theory and Its Applications*, John Wiley & Sons, Inc., 1960.

10. Patrick, E.A., *Fundamentals of Pattern Recognition*, Prentice-Hall, Inc., 1972.

11. Wilensky, R., *LISPcraft*, W.W. Norton & Company, 1984.

## INITIAL DISTRIBUTION LIST

| | | No. Copies |
|---|---|---|
| 1. | Defense Technical Information Center<br>Cameron Station<br>Alexandria, VA 22304-6145 | 2 |
| 2. | Library, Code 0142<br>Naval Postgraduate School<br>Monterey, CA 93943-5002 | 2 |
| 3. | Chief of Naval Operations<br>Director, Information Systems (OP-945)<br>Navy Department<br>Washington D.C. 20350-2000 | 1 |
| 4. | Department Chairman, Code 52<br>Department of Computer Science<br>Naval Postgraduate School<br>Monterey, CA 93940-5000 | 2 |
| 5. | US Army Combat Developments<br>Experimentation Center (USACDEC)<br>ATTN W.D. West<br>Fort Ord, CA 93941 | 2 |
| 6. | Artificial Intelligence Center<br>HQDA OCSA<br>ATTN: DACS-DMA<br>The Pentagon RM 1D659<br>Washington D.C. 20310-0200 | 1 |
| 7. | Curriculum Officer, Code 37<br>Computer Technology<br>Naval Postgraduate School<br>Monterey, CA 93940-5000 | 1 |
| 8. | Professor Robert B. McGhee, 52MZ<br>Computer Science Department<br>Naval Postgraduate School<br>Monterey, CA 93940-5000 | 3 |
| 8. | Professor Michael J. Zyda, 52ZK<br>Computer Science Department<br>Naval Postgraduate School<br>Monterey, CA 93940-5000 | 2 |

10. Col.& Mrs. R.W. Sando (USA Ret.) 1
2602 Autumn Green Drive
Orlando, FL 32822

11. LT Jean M. Sando 1
2602 Autumn Green Drive
orlando, FL 32822

12. United States Military Academy 1
Department of Geography & Computer Science
ATTN: CPT Mark Davis
West Point, NY, 10996-1695

13. LT Tom Wetherald 1
3412 Hunts Point Rd.
Bellevue, WA 98004

14. Director of Research Administration 1
Code 012
Naval Postgraduate School
Monterey, CA 93943

END

DATE

FILMED

MARCH

1988

DTIC